Verifiable delegated authorization for user-centric architectures and an OAuth2 implementation

Luca Ferretti, Mirco Marchetti, and Michele Colajanni
University of Modena and Reggio Emilia
Email: {luca.ferretti, mirco.marchetti, michele.colajanni}@unimore.it

***Abstract*— Delegated authorization protocols have become wide-spread to implement Web applications and services, where some popular providers managing people identity information and personal data allow their users to delegate third party Web services to access their data. In this paper, we analyze the risks related to untrusted providers not behaving correctly, and we solve this problem by proposing the first verifiable delegated authorization protocol that allows third party services to verify the correctness of users data returned by the provider. The contribution of the paper is twofold: we show how delegated authorization can be cryptographically enforced through authenticated data structures protocols; we extend the standard OAuth2 protocol by supporting efficient and verifiable delegated authorization including database updates and privileges revocation.**

***Index Terms*—integrity; oauth; identity; correctness; outsourcing; authorization; access control; cloud**

I. Introduction

In modern Internet-based services, few popular data aggregator services, such as Facebook and Google, have acquired large volumes of users data [12], [21] so that several independent third parties may provide customized services by accessing user data stored by data aggregators. In this scenario, data aggregators maintain users identity and authentication information, and allow users to authorize third parties to access to a subset of their data. Mechanisms to regulate data access are based on delegated authorization protocols that overcome the limitations of traditional authorization approaches. The de-facto standard for delegated authorization in Web-based services is OAuth version 2.0 (OAuth2) [15]. All existing solutions for delegated authorization assume service providers to trust correct behavior by data aggregators. For example, service providers cannot verify that data aggregators return all data that users intended to share (complete), nor that these data are correct (authentic) and updated (fresh). Users and services currently trust data aggregators to behave correctly especially because of the reputation of these companies, but a similar assumption is questionable and might be invalid in future Internet user-centric architectures [4], [5] where small size companies act as storage providers.

This paper has two main contributions. We propose the first delegated authorization protocol that allows service providers to verify data correctness returned by data aggregators according to the authorization policy delegated by the users, including data authenticity, completeness and freshness. These guarantees are achieved by reducing delegated authorization protocols to set operations among attributes, and leveraging cryptographic authenticated data structure protocols specialized for efficient verifiable set operations. The second contribution is a candidate implementation of the proposed protocol as a backwards compatible extension to OAuth2.

The paper is organized as following. Section II describes the delegated authorization scenario and the threat model. Section III gives the background knowledge related to OAuth2 and authenticated data structure protocols. Section IV describes the novel verifiable delegated authorization protocol and its implementation as an extension to OAuth2. Section V discusses related work. Finally, Section VI concludes the paper.

II. System and threat models

We consider the user-centric model including three roles: *user*, *storage provider* and *service provider*, as shown in Figure 1. A user is an entity (e.g., a person) that generates contents (*users data*). A storage provider (or data aggregator) offers outsourced data storage and retrieval services. Users externalize their data to these storage providers that guarantee high availability and performance. A service provider is a third-party that offers Web-based applications and services to users. In such a way, service providers require access to the portion of users data that is relevant to their services, and that users are willing to share with them. Service providers access users data through a delegated authorization protocol. We assume that users have private *user credentials*, enabling them to access storage and service providers.

The considered delegated authorization model is based on attribute-based access control [14]. Each element of user data is associated with one or more *attributes* that are defined by the users. As an example, possible attributes of a digital images storage provider may include *picture*, *drawing*, *public*, *private*, *family*, *friends*. A user uploading a digital picture of his family may associate it with the attributes *picture* and *family*. Users also define *access control rules* for delegated authorization specifying which data can be accessed by a service provider. Each access control rule is a Boolean formula defined over the available attributes. The service provider can access all and only user data satisfying his access control rules. Following the previous example, the user grants a service provider with the access rule "*picture* `and` (*family* `or` *public*)". When the service provider requires user data, the storage provider sends him all the images whose attributes satisfy the access control formula. We highlight that this scenario supports dynamic

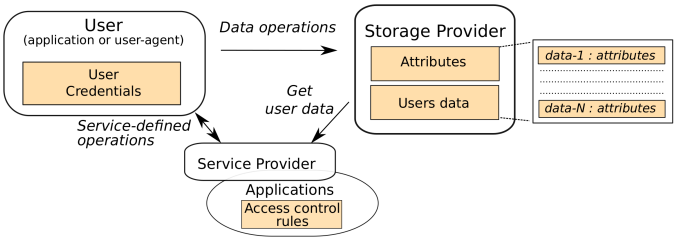

Fig. 1. Delegated authorization model

databases, where users execute all CRUD (create, read, update, delete) operations [7], and change the associated attributes. Users can also grant and revoke access control rules to their service providers at any time.

This scenario faces many security threats regarding data at rest, in motion and in use. Many attacks can be avoided by using standard cyber defenses. We assume that all parties communicate through the Internet by using standard protocols. Similarly to OAuth2, our proposal adopts HTTPS to guarantee security of data in motion. Moreover, we assume that the storage and service providers have valid digital certificates generated by trusted Certification Authorities that allow other parties to verify their identity. Providers might also protect data at rest using standard cryptographic techniques, such as transparent disk or database encryption. These techniques are orthogonal to the proposed protocol.

More interesting attacks still representing open research issues are related to adversarial storage providers. The solution proposed in this paper considers a threat model in which a storage provider may: refuse access to authorized service providers; return data not complying with the access rule of a service provider; return only a subset of all data complying with the access rule; return data not belonging to the user, such as data with arbitrary modifications; return old versions of deleted or updated users data.

III. Base knowledge

A. OAuth2 framework for Web applications

The OAuth2 framework covers many use cases, where the client role is instantiated by different devices and managed by different entities, such as a mobile device managed by the resource owner or a Web application managed by a third party [15], [16]. This paper focuses on OAuth2 authorization protocols for Web-based applications.

OAuth2 enables an application to obtain limited access to resources stored on a Web service on behalf of their owner. The standard [15] identifies four roles: *client*, *resource owner*, *authorization server*, and *resource server*. In the delegated authorization model presented in Section II, *clients* correspond to *service provider applications* and *resource owners* correspond to *users*. *Authorization server* and *resource server* are components included in the *storage provider*. The resource server stores users data and distinguishes between authorized and unauthorized access requests. The authorization server issues *bearer tokens* to the service provider application after a successful authorization.

Roles, data and main information flows of OAuth2 are shown in Figure 2. A service provider application registers at the storage provider to obtain valid credentials, including a *client_id* identifier, and to negotiate access to a set of attributes (*allowed scopes*). Before accessing user data, the service provider application starts an *authorization flow* to obtain *bearer tokens* from the storage provider. A bearer token is associated with the *granted scopes* defined by the user as an access control rule. For Web applications, the authorization flow used by OAuth2 is the *authorization code flow*. We outline the most important messages of this protocol in Figure 3.

A user intending to delegate access to some of his data to a service provider application receives a Web page including a *URI* that points to the storage provider authorization endpoint, a *scope* field containing the required access control rule, and the *client_id* identifying the service provider application. The user sends the *scope* and the *client_id* to the *URI*. This request is authenticated through the *user credentials*. The storage provider presents a confirmation form showing the scope information that the user accepts. Then, the storage provider redirects the user to the OAuth2 callback endpoint of the service provider application including a short-lived *authorization_code* and the *scope*. The service provider application forwards the *authorization_code* to the storage provider, authenticating itself through its *service credentials*. The storage provider replies with long-lived *bearer tokens*. The service provider application can use these tokens to access *users data* in later interactions with the storage provider.

B. Authenticated data structures framework

Authenticated Data Structures [22] enable users that delegate computations to untrusted servers to verify the correctness of the results. Informally, the data owner stores a cryptographic data structure in the untrusted server, that is usually called *the* authenticated data structure (ADS). At the same time, he is able to distribute a small size cryptographic digest to clients that will query his data. The ADS allows clients to verify correctness of computation against the digest. For any query, the server must return some proof of correct computation based on the ADS. The client can detect whether the provider manipulates the ADS or the proof thanks to cryptographic security guarantees. The ADS framework formally describes this approach through five algorithms:

- *Keygen* is a probabilistic algorithm executed at the beginning of the protocol to produce private and public keys;
- *Setup* allows the data owner to initialize cryptographic data structures;
- *Update* allows the data owner to update cryptographic data structures;
- *Query* allows the server to produce a proof of computation for a query;
- *Verify* allows a public user to verify that the output provided by the server is correct, including authenticity, completeness and freshness.

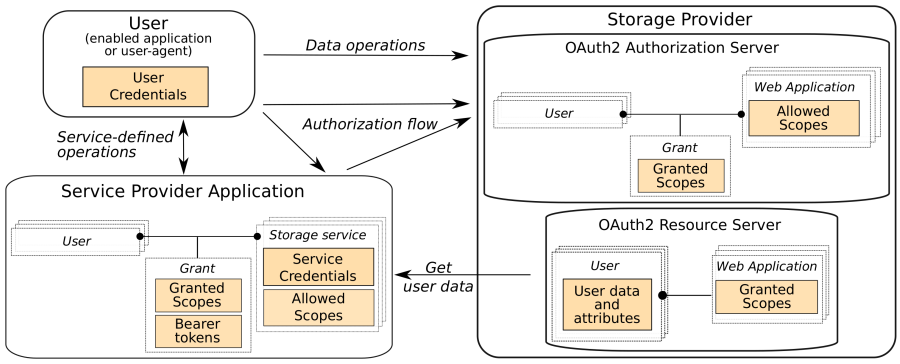

Fig. 2. OAuth2 architecture in the user-centric scenario

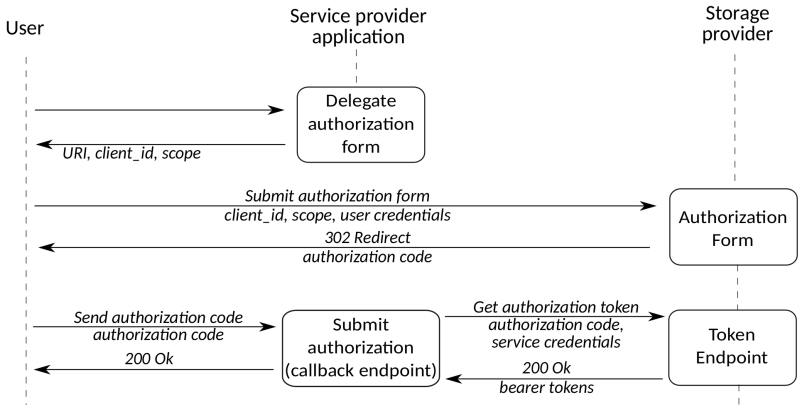

Fig. 3. OAuth2 workflow for Web Applications

In this paper we consider ADS protocols that enable efficient verifiable set operations in the memory checking model. In this setting, the database is represented as a memory (also, a set collection) where each *cell* is a set of values identified by a unique *label*. The protocol allows the infrastructure to verify the correctness of queries where results are the output of *set operations* (union, intersection, disjunction) over sets of values stored in the memory cells (also, the sets of the set collection) identified by the *labels*. Our scenario requires public verifiability, hence we only focus on protocols based on public-key cryptography [3], [10], [18].

IV. Verifiable delegated authorization protocol

This section presents the first protocol for verifiable delegated authorization. The protocol is independent of any solution for delegated authorization. Since the considered scenario can be immediately mapped to an instance of the OAuth2 framework, without any loss of generality, we describe our protocol as an extension to OAuth2, but it can be generalized and applied to other protocols.

In the proposed protocol all parties maintain and exchange additional cryptographic information and metadata. It does not require additional trusted third-parties, roles or components. All data are exchanged through messages and fields of the original OAuth2 framework that are extended to support the new security guarantees while maintaining backward compatibility. The protocol can be instantiated by using any cryptographic protocol that is compliant with the ADS framework. In the current version, we describe it by referring to the proposals for efficient verifiable set operations [3], [10], [18]. We first describe the new data structures (Section IV-A). Then, we present the extended protocol operations (Section IV-B) and discuss the additional security guarantees (Section IV-C).

A. Data structures

Figure 4 describes the extended version of OAuth2 supporting verifiable delegated authorization. The additional data structures are: *secret and public keys*, *grant attestation*, *revoke attestation*, *user data digest (Digest)*, *authenticated data structure (ADS)*.

Secret and public keys are cryptographic keys of a user. The secret key is generated and maintained only by the user. The public key is computed by the user and distributed to the storage and service providers. They include key pairs required by verifiable set operations and digital signatures algorithms. *Grant attestation* is a digital certificate of an access rule granted by the user to the service provider application and signed by the storage provider. This information allows the service provider application to demonstrate compliance to an authorization rule granted by the user. *Revoke attestation* is a digital certificate assessing that a user revoked an access rule previously granted to a service provider application. *Digest* is a small-sized cryptographic material representing the state of the user data. The user updates this information each time

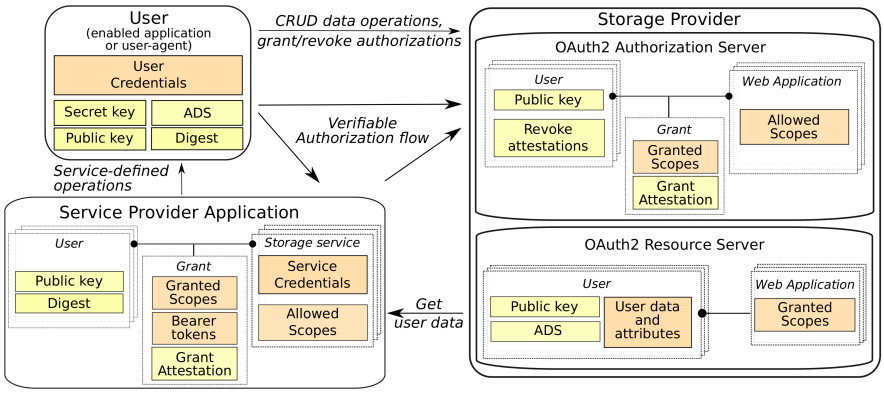

Fig. 4. Architecture for verifiable delegated authorization

he updates his data. Moreover, the service provider application uses this information to verify freshness of the results returned by the storage provider. *Authenticated data structure* is a cryptographic data structure associated with the user data that support efficient cryptographic algorithms for verifiable set operations. Each user maintains his own ADS and the storage provider stores an ADS for each user. The size of the ADS is asymptotically logarithmic with respect to the cardinality of the set of *labels* that define the access control rules, and constant with respect to the amount of data stored by the storage server (see [3], [18] for details).

B. Protocol operations

The proposed protocol defines five operations: *registration*, *verifiable authorization flow*, *user data operations*, *revocation*, and *verifiable application operations*. We use the notation described in Section III to refer to the ADS framework algorithms: *setup*, *update*, *query* and *verify*.

Registration is a one-time interaction between parties to setup the system before running the protocols. We distinguish between *(a)* users and *(b)* service providers registrations. *(a)* A user registers at the storage and service providers. He obtains user credentials and sends his *public key* to the providers. The user gets the allowed attributes by the storage provider and runs the *setup* routine to initialize the ADS, using the attributes as the labels of the authenticated set collection. Moreover, he sends the initial ADS to the storage provider and the initial Digest to the service provider application. *(b)* A service provider registers an application at the storage provider to obtain *service credentials*, including the OAuth2 *client_id*, and to negotiate the portions of the users data he needs to access (the *allowed scopes*).

User data operations include all CRUD and privilege modification operations allowing a user to manage his data. When the user updates his data, he runs the *update* routine to produce cryptographic material that allows the storage provider to efficiently modify its ADS. Each new element of data inserted in the storage service is also inserted in the authenticated set collection for all the attributes associated with it. As an example, if the user stores an image with attributes *picture* and *family*, he inserts the image at locations *picture* and *family* of the authenticated set collection. Similarly, to remove data the user updates the authenticated set collection by removing it from all locations corresponding to the associated attributes. The user can also add or remove attributes from the data by adding and removing it just from the locations associated to the added or removed attributes. These operations do not modify the space requirements of ADS because, as mentioned above, the size of the ADS is constant with regard to the amount of records stored in each set of the set collection.

The *verifiable authorization flow* allows a service provider application to obtain bearer tokens to access user data on the storage provider. We design the protocol as a variant of the *authorization code* flow of OAuth2 (see Figure 3). The protocol goal is to let the storage provider and the service provider application agree on an access control rule that is granted by the user. They exchange a grant attestation that proves the agreement and protects both sides against each other's false claims. We describe this protocol by referring to Figure 5. As in the original OAuth2, a user wishing to use some functions on a service provider application receives a Web page including the *URI* of the storage provider OAuth2 authorization endpoint, a *scope* field with the required access control rule, and the *client_id* identifying the service provider application. Moreover, it receives the *state* field including a nonce that identifies this flow and the signature of the service provider computed over the nonce and the value of *scope*. The user follows the URI transmitting the *scope*, the *client_id*, the *response type* and the *state* values to the storage provider. This request is also authenticated by using the *user credentials*. The storage provider presents a confirmation form showing the scope information that the user accepts. After user acceptance, the storage provider redirects the user to the OAuth2 callback endpoint of the service provider application, including a short-lived *authorization_code*, along with *scope*. The service provider application forwards the *authorization_code* to the storage provider, authenticating itself through its *service credentials*. The storage provider replies with *bearer tokens* and the *grant attestation*. The *grant attestation* certifies that the storage provider and the service provider agreed on the access control rule included in the *scope* field.

Revocation allows users to revoke an access control rule granted to a service provider application. The user needs

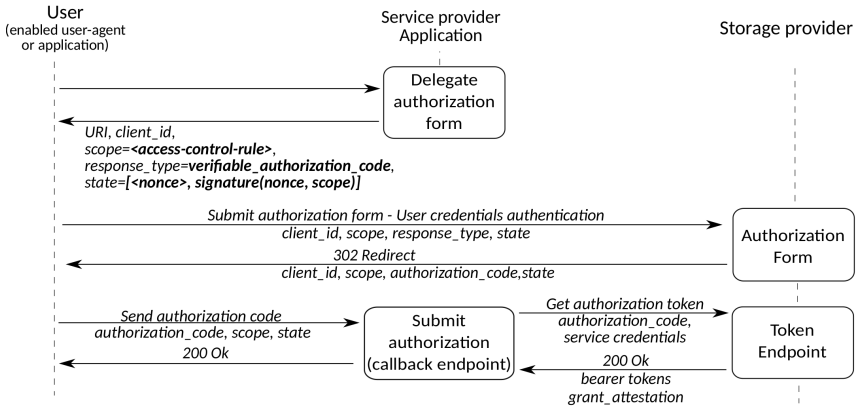

Fig. 5. Verifiable authorization code for extended OAuth2

the access rule identifier, that is the *nonce* generated by the service provider application and included in the *state* field of the *verifiable authorization flow* previously used to grant the access control rule. The user can store this value locally when executing the flow or retrieve it from the storage provider. To issue a revocation, the user produces a *revoke attestation*: a digitally signed revocation certificate through the user key including the *nonce*. The storage provider maintains a copy of the *revoke attestation* to reject requests from service provider applications that try to access users data using the revoked access rule.

Verifiable application operations are verifiable data retrieval operations issued by a service provider application to the storage provider. We assume that the service provider application previously obtained bearer tokens associated to the access control rule authorizing it to retrieve a subset of users data. Let us assume that the user issues a request to the service provider application that requires access to some of his data. Within the request, the user includes the current version of the user *Digest* (e.g., within an opaque header in the HTTP request). The service provider identifies the user data required to satisfy the request and the related access rule. Then it requests users data to the storage provider by submitting the *access rule* and its *bearer tokens*. The storage provider verifies this query by checking if a granted access control rule exists and is still valid for the received bearer tokens. If the access control rule is valid, the storage provider retrieves the users data satisfying the rule. Moreover, he uses the rule as input of the *query* algorithm on the ADS. The output is a cryptographic proof demonstrating correctness of retrieved data. The storage provider returns users data and the proof to the service provider application, that uses the proof to run the *verify* algorithm against the *access control rule* and the *Digest*. If the *verify* algorithm is successful, the results returned by the storage provider are correct.

C. Security guarantees

Let us consider two scenarios: the storage provider accepts the query of the service provider application; the storage provider refuses the query of the service provider declaring that the service is not authorized.

In the first scenario, the correctness of results produced by the storage provider relies on the security guarantees of the ADS protocol. If the *verify* algorithm succeeds, the service provider applications knows that users data returned by the storage provider are complete, authentic and fresh. If the *verify* algorithm fails, the service application provider knows that at least one of these properties is not satisfied, hence the storage provider did not behave correctly. Moreover, since the proposed protocol for verifiable delegated authorization is built on verifiable set operation primitives that are publicly verifiable, the service provider can use the cryptographic proof generated by the storage provider to demonstrate his incorrect behavior to any third party.

In the second scenario, a correct service provider application can demonstrate its rights to access the required portion of users data by exhibiting his *grant attestation*. A correct storage provider can refuse to fulfill the query only if the *access control rule* that delegates data access to the service provider application has been revoked. In this case, the storage provider replies with the *revoke attestation* that is signed by the users and verifiable by any third party. On the other hand, an adversarial storage provider that arbitrarily refuses to fulfill a correct query issued by an authorized service provider application does not own the *revoke attestation* for the *access control rule* authorizing data access. Hence, the *revoke attestation* cannot be included in its response. This security guarantee relies on the unforgeability of the digital signature produced by the user and included in *revoke attestations*.

V. Related work

The literature most related to the proposed protocol regards cryptographic schemes for access control enforcement and verifiable computation, and architectures for securing delegated authorization scenarios. Attribute-based encryption [2], [13], [6], key distribution schemes and property-preserving encryption [9], [11], [8] guarantee confidentiality of out-

sourced data in presence of fine-grained access control requirements. Although many of these schemes guarantee also data authenticity, they do not allow to verify correctness. As a result, their application in the delegated authorization field cannot guarantee completeness and freshness of results, that are novel features of the verifiable delegated authorization protocol proposed in this paper.

Digital signature schemes represent the standard approach to publicly demonstrate authenticity of data, but cannot be adopted as-is to produce proofs of correct computation. Proposals exist to efficiently guarantee integrity of outsourced data in presence of complex queries, including verifiable operations in cloud databases [1], [20]. However, these schemes do not fit the query model of delegated authorization scenarios where the inputs of the computations are not known a priori. Cryptographic schemes have been designed to enforce verifiable computation on any computation [19]. However, their computational overhead depends on the size of the operations implemented as binary circuits, making their feasibility for complex set operations questionable. On the other hand, we consider specialized verifiable set operations protocols that support efficient verification of set operations [3], [10], [18]. To the best of our knowledge, these works have never been applied to the field of delegated authorization.

An alternative approach to improve security of data stored in data aggregators infrastructures is proposed in [17]. This paper adopts software and hardware data isolation strategies within the provider's architecture. It represents an effective approach to improve data security against external attackers, but it cannot guarantee users against malicious providers. The interesting proposal in *Sieve* [23] protects confidentiality of data stored by data aggregators against data breaches, enforces end-to-end confidentiality between users and applications, but it does not allow service providers to verify the correctness of data returned by data aggregators. Guaranteeing confidentiality of outsourced data is out of the scope of this paper, however integrating similar feature with our proposal is an interesting future work.

VI. Conclusions

This paper presents a novel protocol that guarantees verifiability of delegated authorizations. The main idea is to extend existing architectures with cryptographic authenticated data structures that enable efficient verifiable sets operations protocols. By adopting this cryptographic primitive, service provider applications can verify the correctness of all results returned by storage services, including completeness, authenticity and freshness. These security guarantees also hold in dynamic database scenarios in which the data owner can create, update and delete outsourced data, add or remove attributes and issue or revoke access control rules at any time. The protocol is described through a prototype implementation extending OAuth2 that is the de-facto standard for delegated authorization in Web services. In future work we aim to extend this research to guarantee even data confidentiality, and to implement a campaign of performance evaluation under realistic workloads.

References

[1] A. Andreoli, L. Ferretti, M. Marchetti, and M. Colajanni. Enforcing correct behavior without trust in cloud key-value databases. In *Second IEEE Int. Conf. Cyber Security and Cloud Computing*, 2015.

[2] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proc. 2007 IEEE Symp. Security and Privacy*.

[3] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos. Verifiable set operations over outsourced databases. In *Proc. 2014 Int. Workshop Public Key Cryptography*. Springer LNCS.

[4] T. Chajed, J. Gjengset, J. van den Hooff, M. F. Kaashoek, J. Mickens, R. Morris, and N. Zeldovich. Amber: Decoupling user data from web applications. In *Proc. 15th USENIX Workshop Hot Topics in Operating Systems*, 2015.

[5] R. Chandra, P. Gupta, and N. Zeldovich. Separating web applications from user data storage with bstore. In *Proc. 2010 USENIX Conf. Web application development*.

[6] M. Chase. Multi-authority attribute based encryption. In *Proc. 2007 Conf. Theory of Cryptography*. Springer LNCS.

[7] L. Ferretti, M. Colajanni, and M. Marchetti. Supporting security and consistency for cloud database. In *Proc. 4th Int'l Symp. on Cyberspace Safety and Security*, 2012.

[8] L. Ferretti, M. Colajanni, and M. Marchetti. Access control enforcement on query-aware encrypted cloud databases. In *Proc. 5th IEEE Int. Conf. Cloud Computing Technology and Science*, 2013.

[9] L. Ferretti, M. Colajanni, and M. Marchetti. Distributed, concurrent, and independent access to encrypted cloud databases. *IEEE Trans. Parallel and Distributed Systems*, 25(2), 2014.

[10] L. Ferretti, M. Colajanni, and M. Marchetti. Implementation of verified set operation protocols based on bilinear accumulators. In *Proc. 15th Int. Conf. Cryptology and Network Security*, 2016.

[11] L. Ferretti, F. Pierazzi, M. Colajanni, and M. Marchetti. Performance and cost evaluation of an adaptive encryption architecture for cloud databases. *IEEE Trans. on Cloud Computing*, 2(2), 2014.

[12] P. Gill, V. Erramilli, A. Chaintreau, B. Krishnamurthy, K. Papagiannaki, and P. Rodriguez. Follow the money: understanding economics of online aggregation and advertising. In *Proc. 2013 ACM Conf. Internet measurement conference*, 2013.

[13] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. 13th ACM Conf. Computer and communications security*, 2006.

[14] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, et al. Guide to attribute based access control: definition and considerations. *NIST Special Publication*, 800(162), 2013.

[15] IETF. RFC 6749: The oauth 2.0 authorization framework. https://tools.ietf.org/html/rfc6749, last visited May 2017.

[16] IETF. RFC 6819: Oauth 2.0 threat model and security considerations. https://tools.ietf.org/html/rfc6819, last visited May 2017.

[17] J. Kannan, P. Maniatis, and B. G. Chun. Secure data preservers for web services. In *Proc. Second USENIX Conf. Web application development*, 2011.

[18] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. Trueset: Faster verifiable set computations. In *Prof. 23rd USENIX Symp. Security*, 2014.

[19] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Proc. 2013 IEEE Symp. Security and Privacy*.

[20] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling security in cloud storage slas with cloudproof. In *Proc. USENIX Annual Technical Conference*, 2011.

[21] C. Riederer, V. Erramilli, A. Chaintreau, B. Krishnamurthy, and P. Rodriguez. For sale: your data: by: you. In *Proc. 10th ACM Workshop Hot Topics in Networks*, 2011.

[22] R. Tamassia. Authenticated data structures. In *Prof. 2003 European Symposium on Algorithms*. Springer LNCS.

[23] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan. Sieve: cryptographically enforced access control for user data in untrusted clouds. In *13th USENIX Symp. Networked Systems Design and Implementation*, 2016.