An Architecture for Trustworthy Open Data Services

Andrew Wong, Vicky Liu, William Caelli, and Tony Sahama

Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia.
(jianwye.wong, v.liu, w.caelli, t.tony)@qut.edu.au

Abstract. This paper addresses the development of trust in the use of Open Data through incorporation of appropriate authentication and integrity parameters for use by end user Open Data application developers in an architecture for trustworthy Open Data Services.The advantages of this architecture scheme is that it is far more scalable, not another certificate authority hierarchy with massive dispersion of key certificates which has of late become too widespread and unmanageable. With the use of a Public Key File, if the key is compromised; it is a simple matter of the single responsible entity replacing the key pair with a new one and re-performing the data file signing process. Responsibility for Open Data is separated from any other certificate authority that might be used by the publishing entity. Under this proposed architecture, the Open Data environment does not interfere with the internal security schemes that might be employed by the entity. However, this architecture incorporates, when needed, parameters from the entity, e.g. person who authorized publishing as Open Data, at the time that datasets are created/added.

Keywords: Open Data, Integrity, REST, Security, Public File

1 Introduction

During the course of his doctoral study, Roy Fielding generalized the architectural principles that drove the Web conceived of by Tim Berners-Lee in the early 1990s and presented these principles as an architectural style which was underpinned by a framework of constraints. This framework was named Representational State Transfer (REST) [1] and systems which adhere to this framework are called "RESTful" systems or services. Because of the REST framework's ease of use and deployment, it has since been used in a variety of other development methodologies, including web services and application programming interface (API) development, and has since become a serious rival to the use of the earlier Simple Object Access Protocol (SOAP) [2] which is a successor to the Remote Procedure Call (RPC) programming style. These SOAP and REST based APIs have been used to communicate data and information in many fields, most recently, Open Data.

Open Data is data that can be freely used, shared and built-on by anyone, anywhere for any purpose [3]. The recent global trends towards Open Data have organizations

and governments relying on these APIs to communicate Open Data in a greater extent than before. The United States, United Kingdom, Japan, and Australia among others, use a software called the Comprehensive Knowledge Archive Network (CKAN), an open source data management platform to manage and publish their Open Data [4]. CKAN's Action API is based on the RPC programming style. Another well-known Open Data Management Suite is Socrata. The Socrata Open Data API (SODA) provides an open, standards-based REST API [5]. In the case of the United States, the wide variety of government services and organizations has led to the use of both CKAN [6] and SODA [7] at the state and federal government levels.

One of the key advantages of Open Data is that it increases the availability of data for consumers in decision making as well as providing potential for massive cost reduction through implicit outsourcing of information system development. At regional governmental level, a combination of Open Data with appropriate interrogation programs could possibly replace physical publication of such documents as guidebooks, listings, etc. The active interest and support by various national and international non-governmental organization as well as state and federal government sponsored 'hackathons' such as GovHack [8] and HealthHack [9] aiming to develop new applications that use Open Data also could lead to a strong upsurge in the use of Open Data in various fields.

As the capacity for acquiring and storing data increasing from year to year and with data analytics exerting greater influence on decision making than in years past, trust has to be placed in not just the processes and algorithms used to analyse data, but in the authenticity and integrity of data as well. There is now a fast developing trend for enterprises; both public and private, to incorporate Open Data with proprietary and private data collections in order to provide better trend and other reports. However, how can users trust conclusions or decisions made on the basis of results obtained from largely, untrusted data with essentially unknown provenance?

An adversary, wishing to use any means necessary to cause disruption, may wish to misuse the Open Data movement to achieve this disruption within the society by:

1. Diverting Open Data requests to a fraudulent site containing fraudulent datasets
2. Insertion of a fraudulent dataset into a legitimate site
3. Deliberate modification of a legitimate dataset
4. Denial of Service should Open Data become an integral and essential part of a community service

Therefore, it becomes of vital significance to ensure the authenticity and integrity of Open Data in order to placing trust in decision making based on that same Open Data, for users, businesses, industry and government alike.

2 Paper Scope

The definition of "Open" in Open Data can be summed up in the statement by the Open Knowledge Foundation that: "Open means anyone can freely access, use, modi-

fy (build upon), and share for any purpose (subject, at most, to requirements that preserve provenance and openness)." [10] Provenance in this case, is taken to mean the authenticity and integrity of data. In keeping with this statement, methods such as encryption which acts to preserve the confidentiality aspect of data may not be completely relevant in the broad philosophy of Open Data, but may be briefly discussed.

There are two aspects to Open Data; 1) the management of Open Data collections, one of which involves actions by an authoritative source like adding, modifying or deleting datasets, and, 2) the usage or modification of datasets post-addition. This paper addresses development of trust in the usage or adaptation of Open Data through the incorporation of appropriate authentication and integrity parameters for data included in end-user Open Data applications by developers. The principle here is that the average person would not normally access raw Open Data collections but would view them through the lens of an appropriate application. The user would therefore need to be able to trust both the authenticity and integrity of data supplied by the application.

The proposed architecture makes use of the Domain Name System Security Extensions (DNSSEC) for host/server verification. However, the full description of DNSSEC functionality lies outside the scope of this paper, and will only be briefly discussed in relation to how it fits into the proposed architecture as a whole. Forthcoming papers may discuss this aspect in more detail. It should be noted that DNSSEC does not use digital certificates but rather a public key hierarchical registry.

The paper will also discuss the proposed architecture's use of a public key hierarchical registry and digital signatures instead of the traditional certificate authority for the authentication of data publishers and integrity of datasets. Future work will include the results of implementation and further testing.

3 Related Work

A search of the Web revealed few academic sources on REST Security in relation to Open Data. In Fielding's thesis [1], REST was designed to provide simplicity of implementation and scalability but has no pre-defined security protection mechanisms [11] when compared to SOAP, which uses the WS-Security [12] standard. In response to this, several authors have recently suggested mechanisms by which to provide security for REST:

Forsberg [13] proposed an approach where content protection was based on keys being delivered to clients via secure session. Forsberg's approach eliminates the need for repeated SSL/TLS encryption of cached content. Forsberg further notes that their solution is adjusted to match better with caching for data that requires confidentiality protection. An approach which used extended HTTP headers to effect extended username tokens was proposed by Dunlu et al. [14] for user authentication. A secondary password for the username token was required in order to avoid leakage of user password. The approach proposed by Serme et al. [15] had some similarities to Dunlu et al. where they used extended HTTP headers, except that Serme's approach uses the HTTP headers to convey digital certificate and encryption information.

Lee and Mehta [16], investigating some of the security threats to REST-based Web Services concluded that although message encryption by HTTPS was a costly protection method, HTTPS-based data transfer was the best method to ensure data confidentiality. Backere et al. [17] states that the best solution to the RESTful security problem, or the one most conforming to RESTful principles, is to differentiate between messages that need to be encrypted and those that do not, that a message is not-modifiable, and that replayed messages be avoided. They propose a login and REST resource access mechanism that leveraged these concepts.

There is a common consensus that it is necessary for appropriate security mechanisms to be employed in REST Web Services, however the means of accomplishing this as well as the security properties to be protected vary from approach to approach. Most of the solutions presented by these authors however, focus on authenticating the user or protecting the confidentiality of information held in RESTful systems. This however, begs the question: Is it necessary to protect the confidentiality of publicly available Open Data, or even to authenticate users of Open Data?

The answer to this question is: Open Data by its very nature is public data, therefore it should be viewable by the public and not restricted by confidentiality mechanisms. With this, authentication of the end user for read-only access to Open Data is not strictly necessary. Having said this, restricting access to the methods that can be used to alter Open Data resources to authenticated entities, such as the original publisher of the data, or other authorized parties is still required. An important quality that was brought up by Backere et al. [17] which relates to Open Data is that messages and content should be unmodifiable, which basically refers to the integrity of Open Data resources and collections even as they are transferred and cached over the Web.

Diffie and Hellman [18] proposed a “public file” which could be used for authentication purposes, requiring only one key to function. And, by making the public file read-only, this enables one personal appearance to authenticate an identity many times over. This public file would work, in principle, and in conjunction with modern public key cryptography, as a central authority, to identify communication from a specific identity, and establish a hierarchy of trust. This approach has proven unpopular when compared with the public key certificate concept proposed by Kohnfelder [19]. The public key certificate authentication scheme based on Kohnfelder’s thesis is now ubiquitous, however, several issues with it have been highlighted by Clarke [20], most notably privacy and the problems with certificate revocation.

Rivest attempts to address the issue of certificate revocation by proposing that proper certificate infrastructure organization can allow a signer to present a collection of certificates as evidence of authenticity [21]. Another paper authored a few years later by McDaniels and Rubin [22] posits that addressing PKI requirements in large, loosely coupled environments using certificate revocation lists is difficult. A web environment based on REST is designed to be a large, loosely coupled environment, as envisioned by Fielding. [1]

4 Proposed Solution

This paper proposes a Trustworthy Architecture for Open Data Systems (Fig. 1.) which would serve as a precaution against tampering, enabling users to know when a particular resource is genuine or has been tampered with, thus augmenting the REST framework. As mentioned in a previous section, the encryption of request-response messages is not as mission-critical for communicating Open Data. However, measures to protect message integrity and authenticity are still required. This approach is also generic enough to be used in APIs of varying styles.

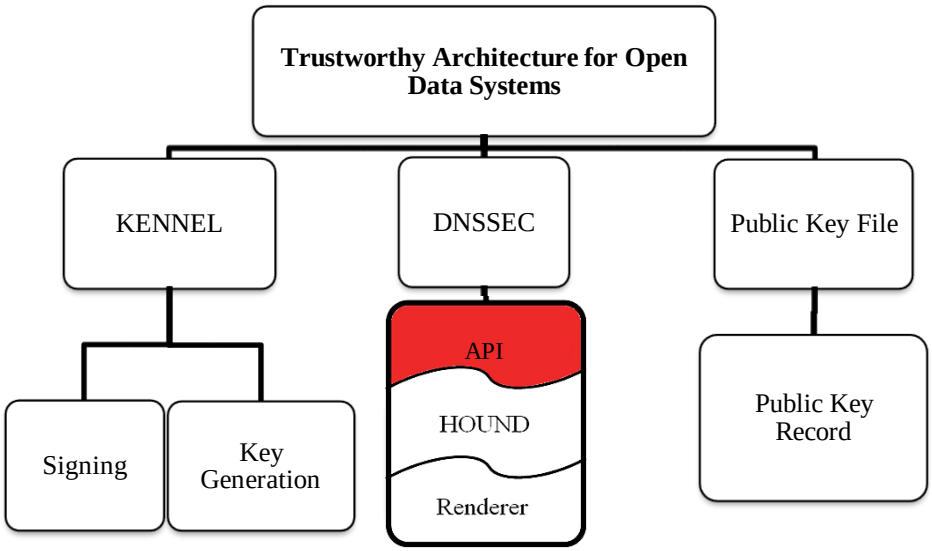

Fig. 1. Trustworthy Architecture for Open Data Systems

4.1 Key Components

The architecture proposed hinges on several key components, a Key Generation component, a Public Key File, DNSSEC and a Verifier Module which interfaces with regular REST framework activity as needed.

Certificate Authority vs Public Key Registry. Effective key management is essential for the smooth operation of cryptographic systems. In regular circumstances, the Trusted Certificate Authority is responsible for issuing digital certificates and maintaining certificate currency and revocation lists, and is also responsible for the generation of cryptographic key pairs and digital signatures. It is the interactions of these two separate components which provide practical key management for practical public key cryptosystems as proposed in Kohnfelder's thesis [19]. If a certificate model is

to be used then this would be the normal procedure, however, given that Open Data is of its own essence, "open", proposing a complex certificate architecture including certificate revocation could be overkill. Moreover, access to confidentiality/privacy services and mechanisms is not required. It would seem reasonable, then, that each Open Data entity could maintain its own public key, relevant to verification of data integrity alone, in an appropriately managed and controlled public key registry (PKR) file similar to the Diffie-Hellman suggested "public file" approach, in line with the philosophy used in DNSSEC.

Public Key Registry. The word "Registry" is used in this case in the sense that it is a central location where public keys associated with recognized identities may be retrieved. It is proposed that any entity wishing to publish Open Data generates a cryptographic key pair and submit the public key to the Open Data Public Key File maintained by this Registry (using the Key Generation and Signing Module). Users of Open Data will then be able to retrieve the appropriate public key to verify a signature from this central location.

Key Generation and Signing Module (KENNEL). For ease of explanation the Key Generation and Signing module will be referred to as KENNEL. To become an Open Data Publisher, owners of Open Data first need to use KENNEL to generate a cryptographic key pair. The module then submits the generated Public Key to the Registry along with sufficient proof of identity over a secure channel (Fig. 2.). This information will be the basis of a record in the Public Key File. (Fig. 3.)

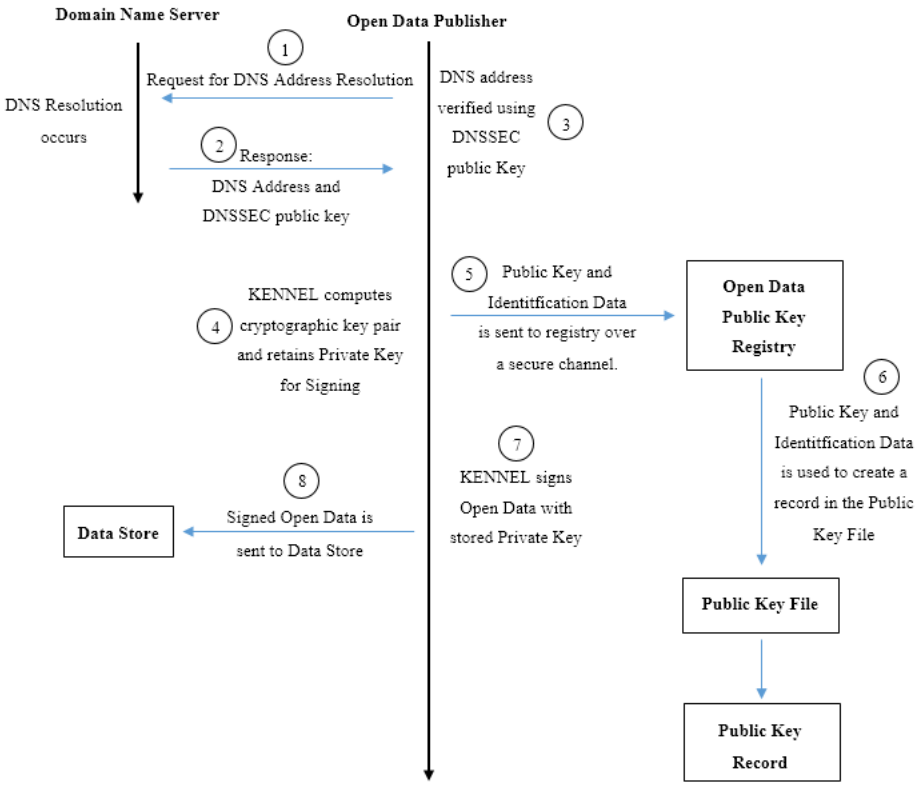

Fig. 2. Open Data Publisher Process using KENNEL

Steps	***Description***
1, 2, 3	DNSSEC enabled Address Resolution
4, 5, 6	KENNEL-Registry interaction: Public Key Record creation
7, 8	KENNEL-Data Store interaction: Open Data signing and storage

Table 1. Process of Fig. 2. Open Data Publisher Process using KENNEL

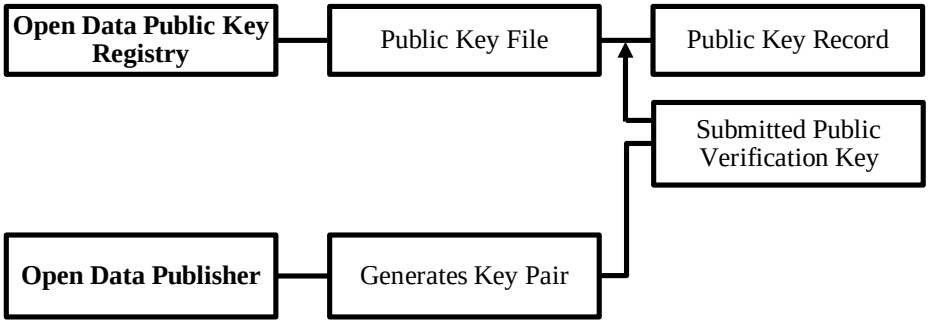

Fig. 3. Role of the Open Data Certificate Authority in relation to the Open Data Publisher

The Open Data Publisher is responsible for maintaining the secrecy of its Private Key, and uses this key in conjunction with a cryptographic one-way hashing function to generate a Digital Signature.

Verifier Module (HOUND). For ease of reference, the Verifier will be referred to as HOUND. After address resolution with DNSSEC implemented is accomplished, a request for resources reaches the server and the server responds with the appropriate resource which is digitally signed. At the client-side, HOUND reads the server response, extracting signer's identity, retrieves the matching public key from its Public Key Record and verifies the digital signature.

A message digest is recovered by decrypting the digital signature with a valid public key and is compared to a message digest computed at the recipient end of the communication. If both digests are identical, then the response is considered to be authentic and retains its integrity. The operation of the verifier terminates and the content is then passed to the Renderer and is displayed in whichever format is applicable. If the digests are not identical, then the recipient is alerted to the fact that the resource may be fraudulent or has been altered in transit.

DNSSEC. The Internet Engineering Task Force has developed RFC3833 [23] the Domain Name System Security Extension specification to resolve various threats to the DNS using public-key cryptography to establish a chain of trust, which in practice means that each DNS records are digitally signed. DNSSEC does not make use of digital certificates but rather a public key hierarchical registry.

In order for DNSSEC to be incorporated, the domain host first needs to have DNSSEC enabled on the server-side and the client needs to install a DNSSEC validator which can read verification information from the server.

4.2 Use Case Scenario

Communication without the proposed architecture. A request for data to a server follows a standard request-response paradigm. This is an example of a standard HTTP request for a resource without the use of the proposed architecture:

```
#REQUEST
GET /dataset/…/streetsandsuburbs HTTP/1.1\r\n
Host: opendataregistry.org
```

Address Resolution is performed after the initial request is made by the Domain Name System. DNS uses Root or Authoritative Name Servers which are heavily supplemented by DNS caches as a workaround to reduce DNS traffic and increase efficiency. Caching DNS request-response records reduces load on individual servers but is vulnerable to DNS cache poisoning and other interception attacks.

```
#RESPONSE
HTTP/1.1 200 OK
Media-Type: JSON
```

After the 200 OK response, there is no further communication from hosting server and there is no provision for integrity verification. Content is then displayed. It is difficult to place confidence in the data because, as mentioned previously, without any security or integrity-checking mechanism in place, the file is still vulnerable to accidental corruption and interception attacks.

Communication using the proposed architecture. The following example illustrates use of the proposed architecture from the perspective of the end user:

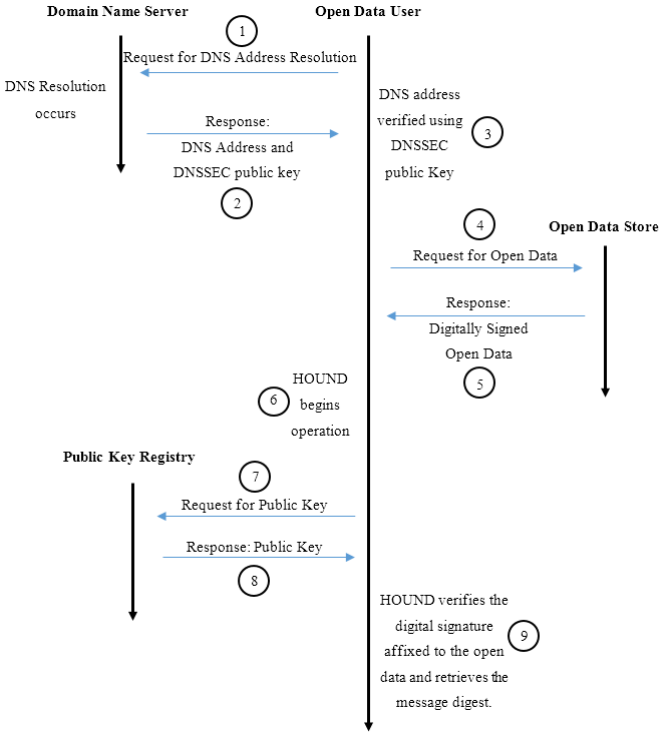

Fig. 2. Open Data User Case Illustration

Steps	*Description*
1, 2, 3	DNSSEC enabled Address Resolution
4, 5	Open Data Resource Lookup
6, 7, 8, 9	HOUND Integrity Verification Process

Table 2. Process of Fig. 4. Open Data User Case

In communication with the end-user, the proposed architecture does not change the request-response format of HTTP communication, but rather augments it with the previously mentioned security components.

```
#REQUEST
GET /dataset/…/streetsandsuburbs HTTP/1.1\r\n
Host: opendataregistry.org
Content-Type: …
```

At this stage, as before, address resolution is performed, incorporating DNSSEC to authenticate the DNS Server and provide defence against cache poisoning or man-in-the-middle attacks. As mentioned in a previous section, a DNSSEC validator is installed on the client-side and validates the incoming content.

After the address resolution is performed successfully, communications proceed as per normal and a response from the hosting server is received. As per RFC2616 [24], a successful request should contain a 2xx or a 3xx status code depending on the HTTP method used. If an error on the client-side is perceived, the server should return a 4xx status code but if an error occurs on the server-side a 5xx status code should be returned to the client.

```
#RESPONSE
HTTP/1.1 200 OK
Media-Type: JSON
Content-Length: …
```

In this case, the response contains a status code and the actual message which is digitally signed and accompanied by a digital certificate. The verifier module is then called to verify the integrity and authenticity of the received message. At the end of verifier module process, the message or content should be displayed if both the certificate and signature pass verification. The following is pseudocode describing the functioning of the verifier module:

```
#VERIFIER-PSEUDOCODE
GET DownloadedContent
READ Digital_Sign from DownloadedContent
EXTRACT KeyFileNumber from Digital_Sign

GET PublicKeyFile matching KeyFileNumber
RETRIEVE PublicKey from PublicKeyFile
```

```
CALL  Decrypt_Sign with PublicKey and Digital_Sign
      RETURNING decrypt_result
STORE decrypt_result in hash1

CALL  hash_compute with DownloadedContent
      RETURNING hash_result
STORE hash_result in hash2

IF hash1 = hash2 THEN
  SHOW message: integrity and authenticity verified
  DISPLAY DownloadedContent
ELSE
  SHOW message: failed integrity check
  TERMINATE
END IF
END
```

This procedure assumes DNSSEC is enabled. However, should DNSSEC not be available, HOUND should use HTTPS as a minimum for integrity to retrieve the appropriate public key from the Public Key File at the Registry for signature verification.

REST URI Interface with proposed architecture. Each dataset may contain one or more resource files, which is linked to a particular digital signature file and is associated with the identity and public key of the dataset owner/creator at the Registry.

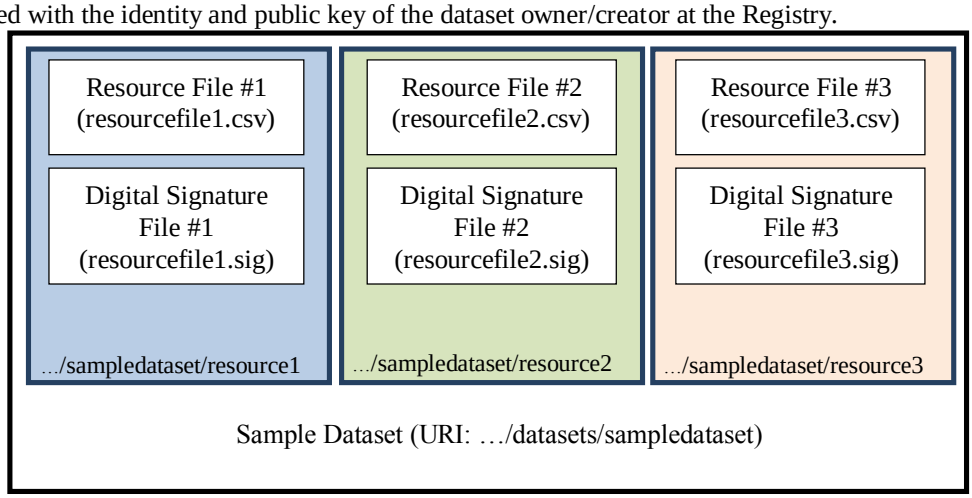

Table 3. URI interface

When a GET request is called for a dataset, e.g. GET /datasets/sampledataset, the server returns a listing of resource files and their URIs. From the information in that list, a GET request may then be sent for an individual resource, e.g. GET

/datasets/sampledataset/resource1, which should retrieve both the resource file and the associated digital signature. All the information is then used in the signature value with the algorithm described in Section 3.2. See Appendix 1 for a previously existing implementation sample of a signature verifier in concrete code.

4.3 Conclusion and Future Work

This paper proposes an architecture that uses digital signatures in conjunction with an associated public key file with the main goal to protect the integrity of Open Data communicated over the Web. This is a simplification of current public key certificate structures which use large revocation lists for certificate currency and have demonstrated problems in scalability and "certificate authority" trust.

The key element of this architecture is that it is based on DNSSEC, which is more appropriate to the new world of IPv6. The advantages of this architecture scheme is that it is far more scalable, not another certificate authority hierarchy with massive dispersion of key certificates which has of late become too widespread and unmanageable. With the use of a Public Key File, if the key is compromised; it is a simple matter of the single responsible entity replacing the key pair with a new one and re-performing the data file signing process.

Responsibility for authenticating Open Data is separated from any other certificate authority that might be used by the publishing entity. Under this proposed architecture, the Open Data environment does not interfere with the internal security schemes that might be employed by the entity. However, this architecture incorporates, when needed, parameters from the entity, e.g. person who authorized publishing as Open Data, at the time that datasets are created/added.

Future work will include the building of a proof-of-concept system using the architecture in this paper and performing benchmarking against regular systems in conjunction with penetration testing. An interesting philosophical question which may be studied in further papers is: what responsibility does an entity, whether private or public, take on when it makes Open Data available? Further study on the issue of Open Data and governance requirements must be done.

5 References

1. Fielding, R. T. "Architectural styles and the design of network-based software architectures" Doctoral dissertation, University of California, Irvine (2000).
2. Don Box, Gopal Kavivaya, Andrew Layman, Satish Thatte, Dave Winer, "SOAP: Simple Object Access Protocol", in Internet Draft draft-box-http-soap-01, November (1999).
3. Defining Open Data (2013) Open Knowledge Foundation Blog. http://blog.okfn.org/2013/10/03/defining-open-data/
4. CKAN instances around the world. http://ckan.org/instances/
5. Socrata Open Data Portal. https://opendata.socrata.com/
6. CKAN Multisite Draft Proposal. https://usopendata.org/2014/12/08/ckan-multisite/
7. Socrata Open Data API. http://www.socrata.com/industries/open-data-state-local-government/

8. GovHack. http://www.govhack.org/
9. HealthHack. http://www.healthhack.com.au/
10. The Open Definition. http://opendefinition.org/
11. Comerford, C., & Soderling, P. “Why REST security doesn't exist?”. http://www.computerworld.com/s/article/9161699/Why_REST_security_ doesn_t_exist
12. OASIS, “Web Services Security: SOAP Message Security 1.1,” (2006). http://www.oasis-open.org/committees/wss
13. Forsberg, D. “RESTful Security.” In Web 2.0 Security & Privacy 2009 in conjunction with 2009 IEEE Symposium on Security and Privacy.(2009)
14. Dunlu Peng; Chen Li; Huan Huo, "An extended UsernameToken-based approach for REST-style Web Service Security Authentication," In Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on , vol., no., pp.582,586, 8-11 Aug. 2009 doi: 10.1109/ICCSIT.2009.5234805
15. Serme, G.; de Oliveira, A.S.; Massiera, J.; Roudier, Y., "Enabling Message Security for RESTful Services," In Web Services (ICWS), 2012 IEEE 19th International Conference on , vol., no., pp.114,121, 24-29 June 2012 doi: 10.1109/ICWS.2012.94
16. Lee, H., & Mehta, M. R. “Defense against REST-based web service attacks for enterprise systems.” In Communications of the IIMA, 13(1), 57-68. (2013). http://search.proquest.com/docview/1518604854?accountid=13380
17. De Backere, F.; Hanssens, B.; Heynssens, R.; Houthooft, R.; Zuliani, A.; Verstichel, S.; Dhoedt, B.; De Turck, F., "Design of a security mechanism for RESTful Web Service communication through mobile clients," Network Operations and Management Symposium (NOMS), 2014 IEEE , vol., no., pp.1,6, 5-9 May 2014 doi: 10.1109/NOMS.2014.6838308
18. W. Diffie and M. E. Hellman, "New Directions in Cryptography," in IEEE Trans. On Info. Theory, Vol. IT-22, Nov. 1976, pp. 644-654
19. Kohnfelder, Loren M. "Towards a practical public-key cryptosystem." PhD diss., Massachusetts Institute of Technology (1978).
20. Clarke, Roger. "Conventional public key infrastructure: An artefact ill-fitted to the needs of the information society." In Proceedings of the 9th European Conference on Information Systems. (2001)
21. Rivest, Ronald L. "Can we eliminate certificate revocation lists?" In Financial Cryptography, pp. 178-183. Springer Berlin Heidelberg, (1998).
22. McDaniel, Patrick, and Aviel Rubin. "A response to “Can we eliminate certificate revocation lists?”." In Financial Cryptography, pp. 245-258. Springer Berlin Heidelberg, (2001).
23. RFC3833. https://tools.ietf.org/html/rfc3383
24. RFC2616. https://www.ietf.org/rfc/rfc2616.txt
25. Code Examples – Verifying Digital Signatures. https://www.flexiprovider.de/examples/E-xampleSMIMEverify.html

6 Appendix 1 – Open Source Java Signature Verifier Implementation by Technische Universität Darmstadt [25]

```
//Imported necessary dependencies
01 import java.io.File;
02 import java.io.FileInputStream;
03 import java.security.MessageDigest;
04 import java.security.PrivateKey;
05 import java.security.Security;
06 import java.security.Signature;
08 import codec.x509.X509Certificate;
09 import de.flexiprovider.common.util.ByteUtils;
10 import de.flexiprovider.core.FlexiCoreProvider;
//Reading the Message for Verification
18 File file = new File("xxx");
19 byte[] message = new byte[(int) file.length()];
20 FileInputStream fis = new FileInputStream(file);
21 fis.read(message);
22 fis.close();
//Reading the Digital Signature
24 File file = new File("RSASignature.sig");
25 byte[] sigBytes = new byte[(int) file.length()];
26 FileInputStream fis = new FileInputStream(file);
27 fis.read(sigBytes);
28 fis.close();
29
30 File file = new File("CertRSA.cer");
31 byte[] encCertRSA = new byte[(int) file.length()];
32 FileInputStream fis = new FileInputStream(file);
33 fis.read(encCertRSA);
34 fis.close();
…

42 X509Certificate certRSA new X509Certificate(encCertRSA);
43 X509Certificate certCA new X509Certificate(encCertCA);
```

[Code Examples – Verifying Digital Signatures. https://www.flexiprovider.de/exam-ples/ExampleSMIMEverify.html]