International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 3, Issue 4, April 2016

Trustworthy URI: Enhancing the Data's On the Web Reliable and Immutable

[1] A.Amalesh [2] D.Rajesh [3] Sanju Mathew Philip [4] Mrs.D.Lavanya, M.E
[1][2][3] UG student, [4] Assistant Professor
Department of CSE, Loyola Institute of Technology, Chennai, India.
[1] amalesh95@gmail.com [2] rajeshvoguish@gmail.com [3] mathewsanju82@gmail.com
[4] lava2it@gmail.com

***Abstract* — To make digital artifacts (undesired or unintended alteration in data introduced in a digital process by an involved technique and/or technology) such as datasets, code, texts, and images verifiable and permanent. Digital artifacts that are supposed to be immutable, there is moreover no commonly accepted method to enforce this immutability. To solve this problem, we propose trusty URIs containing base 64 encryption values.Base64 encoding can be helpful when fairly lengthy identifying information is used in an HTTP environment. For example, a database persistence framework might use Base64 encoding to encode a relatively large unique id (generally 128-bit UUIDs) into a string for use as an HTTP parameter in HTTP forms or HTTP GET URLs. It makes the contents of the data trustworthy which is sent as a URI to the user and it make sure whether it is trusted or not We show how trusty URIs can be used for the verification of digital artifacts, in a manner that is independent of the serialization format in the case of structured data files such as nanopublications .Our goal is to achieve a data security and make the content present is immutable thereby extending the range of verifiability to the entire reference tree.Even the file with large content it becomes possible to implement in enhancing data's on the web and it is fully compatible with existing standards and protocols**

***Index Terms*—Decentralized systems, data publishing, Semantic Web, linked data, resource description framework, nanopublications**

I. INTRODUCTION

In many areas and in particular in science, reproducibility is important. Verifiable, immutable, and permanent digital artifacts are an important ingredient for making the results of automated processes reproducible, but the current Web offers no commonly accepted methods to ensure these properties. Endeavors such as the Semantic Web to publish complex knowledge in a machine-interpretable manner aggravate this problem, as automated algorithms operating on large amounts of data can be expected to be even more vulnerable than humans to manipulated or corrupted content. Without appropriate counter-measures, malicious actors can sabotage or trick such algorithms by adding just a few carefully manipulated items to large sets of input data. To solve this problem, we propose an approach to make items on the (Semantic) Web verifiable, immutable, and permanent. This approach includes cryptographic hash values in Uniform Resource Identifiers (URIs) and adheres to the core principles of the Web, namely openness and decentralized architecture. It directly follows that trusty URI artifacts are immutable, as any change in the content also changes its URI, thereby making it a new artifact. Again, you can of course change your artifact and its URI and claim that it has always been like this. You can get away with that if the trusty URI has not yet been picked up by third parties, i.e. linked by other resources. Once this is the case, it cannot be changed anymore, because all these links will still point to the old trusty URI and everybody will notice that the new artifact is a different one.

Third, trusty URI artifacts are permanent if we assume that there are search engines and Web archives crawling the artifacts on the Web and caching them. In this situation, any artifact that is available on the Web for a sufficiently long time will remain available forever. If an artifact is no longer available in its original location (e.g. the one its URI resolves to), one can still retrieve it from the cache of search engines, Web archives, or dedicated replication services. The trusty URI guarantees that it is the artifact you are looking for, even if the location of the cached artifact is not trustworthy or it was cached from an untrustworthy source.

II. BACKGROUND

Data Mining

Data Mining is an analytic process designed to explore data (usually large amounts of data - typically business or market related - also known as "big data"

Digital Artifacts

A digital artifact is any undesired or unintended alteration of data introduced in a digital process by an involved technique and/or technology.

Immutable

In object-oriented and functional programming, an immutable object whose state cannot be modified after it is created**.**

URI

In computing, a Uniform Resource Identifier (**URI**) is a string of characters used to identify the name of a resource.

URL

A URL is a specific type of Uniform Resource Identifier (URI), although many people use the two terms

URN

In computing, a Uniform Resource Name (**URN**) is the historical name for a Uniform Resource Identifier (URI) that uses the urn scheme. A URI is a string of characters used to identify a name of a web resource. Such identification enables interaction with representations of the web resource over a network, typically the World Wide Web, using specific protocols.

Blank Nodes:

The support for self-references requires us to transform the preliminary content of a trusty URI artifact into its final version, and we can make use of this transformation to also solve the problem of blank nodes in RDF. A blank node is basically an identifier that is only used in a local scope and for which we do not care to specify a concrete URI. Our approach is to eliminate blank nodes during the transformation process by converting them into URIs. Blank nodes can be seen as existentially quantified varia-bles, which we can turn into constants by Skolemization, i.e., by introducing URIs that have not been used anywhere before. Using the trusty URI with a suffix enumerating the blank nodes, we can create such URIs guaranteed to have never been used before (the artifact code being just a place-holder at first, as above):

http://example.org/r3.RACjKTA5dl23ed7 JIpgPmS0E 0dcU-XmWIBnGn6Iyk8B-U#_1

http://example.org/r3.RACjKTA5dl23ed7 JIpgPmS0E 0dcU-XmWIBnGn6Iyk8B-U#_2

This approach solves the problem of blank nodes for normalization, is completely general (i.e., works on any

ni-URIs:

Our approach is compatible with ni-URIs (see above), and all trusty URIs can be transformed into ni-URIs, with or without explicitly specifying an authority:

ni:///sha-256;5AbXdpz5DcaYXCh9l3eI9ruBosi L5XDU 3rxBcBaUO70

ni://example.org/sha-256;5AbXnpz5AcaYX Ch9l3eI9 ruBosiL5XSU3rxBbBaUO70

The fact that the module identifier is lost does not affect the uniqueness of the hash, but to verify a resource all available modules have to be tried in the worst case. To avoid this, we propose to use an optional argument called module:

ni:///sha-256;5AbXdpz5DcaYXCh7l3eI9ruGosi L5XDU 3rxBbBaUO0?module=RA

III. APPROACH:

We propose here a modular approach, where different modules handle different kinds of content on different con-ceptual levels of abstraction, from byte level to high-level formalisms. Besides that, the most interesting features of our approach are self-references, the handling of blank nodes, and the mapping to ni-URIs.

General Structure:

Trusty URIs end with a hash value in Base64 notation (a specific alphanumeric encoding scheme) preceded by a module identifier. This is an example:

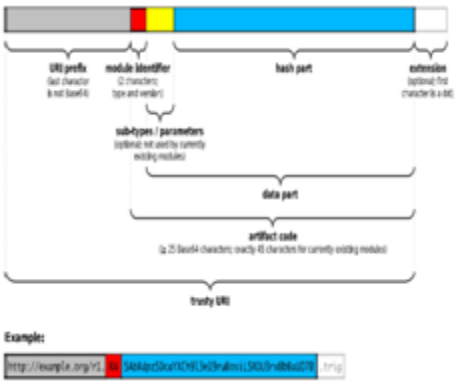


Fig1: URI source process

Architecture:

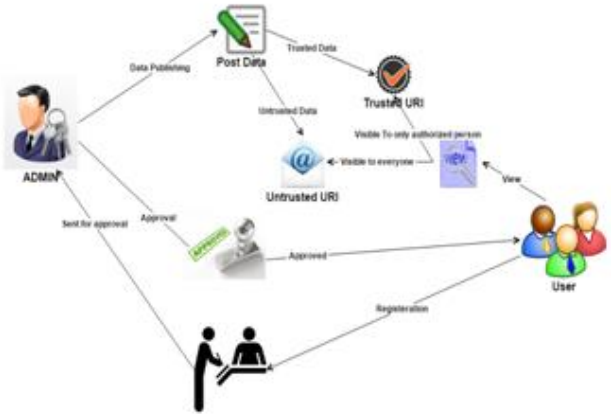


Fig2: Architecture process[

IV. MODULES:

There are currently three modules available: FA, RA, and RB.

Authentication:

Authentication is a process in which the credentials provided are compared to those on file in a database of authorized users' information on a local operating system or within an authentication server. In this project authentication is done to provide more security for the users to have their own credentials to log in.

Authorization:

Authorization is the function of specifying access rights to resources related to information security and computer security in general and to access control. In this project admin approves the users who are registered and provide rights to login to the process.

Cache of the data:

Cache is in wide use and very stable, but has not changed in years and is no longer actively developed. The Cache is designed to assist a developer in persisting data for a specified period of time. In this project it is used as the collection of data to store which is used for various processing.

Encoding:

Encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. The data which is to be published is being encoded and it is been transformed into encoded values and stored it in the database.

Decoding:

Decoding is the opposite process -- the conversion of an encoded format back into the original sequence of characters. The converted data is being decoded back only processing, the Perl implementation makes use of the Trine package, and the Python implementation uses RDFLib.[4] These implementations provide a number of common functions for the different modules and formats. Currently, the following functions are available if the valid user enters in it otherwise it shows that you do not have permission to open the file. CheckFile takes a file and validates its hash by apply-ing the respective module.

ProcessFile takes a file, calculates its hash using mod-ule FA, and renames it to make it a trusty file.

TransformRdf takes an RDF file and a base URI, and transforms the file into a trusty file using a module of type R.Trans form Large Rdf is the same as above but using temporary files instead of loading the entire content into memory.

The trusty URI features provided by the presented libraries are also made available via a validation interface for nanopublications.[6] This interface, which is shown in Fig. 3, offers in fact much more than just validation (including transformation into different formats and publication to nanopublication servers). Users can load nanopublications in different ways, including retrieval from URLs or SPARQL endpoints, and then trusty URIs can be generated for them directly via the Web interface. Nanopublications that already have a trusty URI are automatically verified and users are informed about whether the verification was successful or not.

Publishing the data:

Data publishing is the act of making data available on the Internet, so that they can be accessed, analyzed and reused by anyone for research or other purposes. The data is been published where the appropriate level has the permission to access the file which is determined by admin

V. IMPLEMENTATION:

There are currently three trusty URI implementations in the form of code libraries in Java, Perl, and Python.[3] The Java implementation uses the Sesame library [21] for RDF

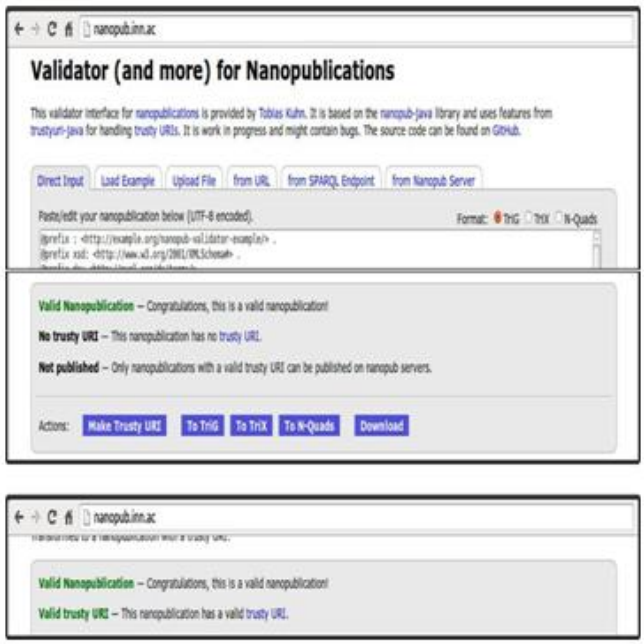


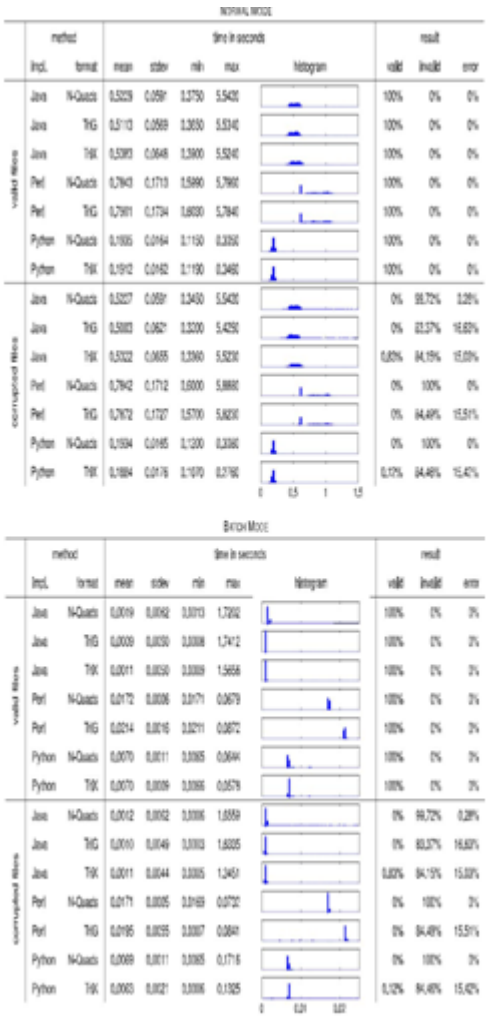
Fig. 3. The nanopublication validator interface integrates features of trusty URIs. Nanopublications can be loaded from different sources, users can generate trusty URIs for them (top), and nanopublications with trusty URIs are automatically verified (bottom).

Hash Generation and Checking on Nanopublications:

To test our approach and to evaluate its implementations, we first took a collection of 156,026 nanopublications in TriG for-mat that we had produced

in previous work [22]. We trans-formed these nanopublications into the formats N-Quads and TriX using existing off-the-shelf converters. Then, we trans-formed these into trusty URI nanopublications using the Java implementation. To be able to check not only positive cases (where checking succeeds) but also negative ones (where checking fails), we made copies of the resulting files where we changed a random single byte in each of them (only considering letters and numbers, and never replac-ing an upper-case letter by its lower-case version or vice.

TABLE 1



versa, as some keywords are not case-sensitive). The resulting six sets of 156,026 files each (three formats, each in two versions: valid and corrupted) were the basis for our evaluation.

VI. PERFORMANCE TESTS ON NANOPUBLICATIONS:

Next, we used the same set of nanopublication files to test the performance of the different modules for checking trusty URI artifacts in different formats. There are two scenarios of how to run such checks: One can run one after the other, as when a small number of nanopublications are manually checked, or one can execute such checks in the form of a batch job in a single program run, which is the preferred procedure to run a large number of checks without supervision. The time required per file is typically much lower in batch mode, as the runtime environment has to start and finalize only once. Therefore it makes sense to have a look at both scenarios.

Table 1 shows the results of these performance checks for the normal mode (top) and batch mode (bottom). These results and the ones presented below were obtained on a Linux server (Debian) with 16 Intel Xeon CPUs of 2.27 GHz and 24 GB of Performance and Results of the Different Implementations for Checking Trusty URI Nanopublications in Normal Mode (Top) and Batch Mode (Bottom) on Valid and Corrupted Files memory. As expected, the time measurements are much lower in batch mode, but checking is reasonably fast also in normal mode. All average values are below 0.8 s (0.03 s for batch mode). Using Java in batch mode even requires only 1 ms per file. Apart from the runtimes, the two modes had no effect on the results.

VII. CONCLUSIONS AND FUTURE WORK:

In conclusion, I think that the first five minutes of an interview are important and can be considered as the first impression that the interviewer gets about you. Because of that, I think that interviews need to focus on these five minutes by following the points described above and in my previous blog posts. If you follow the points I mentioned, you should end up with a great job interview and hopefully the job you applied for.The system has reached a steady state as far as the basicframework is concerned. The system is operated at a high level of efficiency and its advantage is quite understood. Also if time and resource constraints are eliminated, thissystem can be adapted to a full-fledged Knowledge Portal, wherein apersonalized environment for each user whoare a part of it can be created.

REFERENCES:

[1] T. Kuhn and M. Dumontier, “Trusty URIs: Verifiable, immutable, and permanent digital artifacts for linked data,” in Proc. 11th Extended Semantic Web Conf., 2014, pp. 395–410.

[2] P. Groth, A. Gibson, and J. Velterop, “The anatomy of a nano-publication,” Inf. Serv. Use, vol. 30, no. 1, pp. 51–56, 2010.

[3] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker, "Naming things with hashes," Internet Engineering Task Force (IETF), Standards Track RFC 6920, Apr. 2013.

[4] R. Hoekstra, "The MetaLex document server," in Proc. 10th Int. Conf. The Semantic Web, 2011, pp. 128–143.

[5] M. Altman and G. King, "A proposed standard for the scholarly citation of quantitative data," D-lib Mag., vol. 13, no. 3, p. 5, 2007.

[6] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, (2008, Jun.). XML signature syntax and processing. W3C, Recommenda-tion. [Online]. Available: http://www.w3.org/TR/xmldsig-core/

[7] J. Carroll, "Signing RDF graphs," in Proc. 2nd Int. Semantic Web Conf., The Semantic Web, 2003, pp. 369–384.

[8] E. Hofig€ and I. Schieferdecker, "Hashing of RDF graphs and a solution to the blank node problem," in Proc. 10th Int. Workshop Uncertainty Reasoning Semantic Web, 2014, pp. 55.

[9] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryp-tography: The case of hashing and signing," in Proc. 14th Annu. Int. Cryptol. Conf., Adv. Cryptol., 1994, pp. 216–233

[10] C. Sayers and A. Karp, "Computing the digest of an RDF graph," Mobile and Media Systems Laboratory, HP Laboratories, Palo Alto, USA, Tech. Rep. HPL-2003-235(R.1), 2004.

[11] R. Phan and D. Wagner, "Security considerations for incremental hash functions based on pair block chaining," Comput. Security, vol. 25, no. 2, pp. 131–136, 2006