Semantic Insecurity: Security and the Semantic Web

Harry Halpin

Semantic Insecurity: Security and the Semantic Web

Harry Halpin

World Wide Web Consortium/INRIA
2 rue Simone Iff 75012 Paris, France
harry.halpin@inria.fr

Abstract. Strangely enough, the Semantic Web has fallen behind the rest of the Web in terms of security. In particular, we note how TLS is not in use currently for the majority of URIs on the Semantic Web, and how existing Semantic Web standards need to be updated to take into account security best practices. We point out security and privacy flaws in WebID+TLS, and propose alternatives and solutions.

Keywords: security, TLS, WebID, Semantic Web, RDF

1 Introduction

The Semantic Web is based on linked data where items of interest are identified by URIs (Uniform Resource Identifiers) such as *http://www.example.org*. Due to this design choice, to a large extent the Semantic Web crucially depends on these identifiers being able to successfully retrieve documents in order to put the "Web" into the Semantic Web. Strangely enough, the vast majority of these identifiers on the Semantic Web do not use Transport Layer Security (TLS), the widespread IETF standard for encrypting content transmitted over HTTP.[1] This means that documents hosted at these URIs are vulnerable to having their content intercepted and even altered by third parties without the possibility of a user knowing that this is the case, which in turn undermines the fundamental security and trust in the Semantic Web. When a URI is enabled with TLS, the URI uses HTTPS as its scheme in the URI rather than HTTP. TLS was called, and is sometimes still called informally "SSL" (Secure Sockets Layer), and HTTP with TLS enabled adds a "S" for historical reasons to become HTTPS. To be brief, why shouldn't the Semantic Web use *https://example.org* rather than *http://example.org*?

One could claim a Semantic Web URI is only a name and so the usage of TLS is not required. Names in general can be used to refer to things and power logical inferences without retrieving documents. After all, the status of why the Semantic Web uses URIs as opposed to just strings as identifiers is still unclear in the standards. Although there has been much debate from both a

[1] TLS 1.2 is available at *https://tools.ietf.org/html/rfc5246*, with TLS 1.3 being under development at *https://tlswg.github.io/tls13-spec/*.

philosophical and technical perspective about what URIs on the Semantic Web actually refer to, there is not clarity from the core W3C Semantic Web standards for RDF (Resource Description Framework [8]) if Semantic Web URIs should be dereferenced to retrieve data over the Web, much less the security properties these URIs should have [12].

In detail, URIs refer to items of interest, ranging from web-pages *about* the Eiffel Tower to the Eiffel Tower itself [10]. Distinguishing a URI that refers to a document with information about the Eiffel Tower from a URI that denotes the actual iron tower has been relatively difficult, and it has been argued that this kind of ambiguity resolution in knowledge representation is best left to humans as long as the formal semantics that permit inferences are clear [12]. If there is no access to the content of the URI, URIs are the same as any other arbitrary string in a knowledge representation language, and so using HTTP or HTTPS has *no effect* on the formal semantics that determine the inferences that result from a Semantic Web reasoning engine. Yet if this was truly the case then there really is no "Web" in the Semantic Web, and so the use of long HTTP URIs on the Semantic Web is simply an odd naming convention.

The opposing view has also been argued by the Linked Data community that URIs should be linked to actual data that can be retrieved by Semantic Web applications [7]. The original guidelines for "How to Publish Linked Data on the Web"[2] state that URIs for "real world objects" that "exist outside the Web" should be different than the names for "resources we find on the traditional document Web, such as documents, images, and other media files," with this latter kind of resource have been dubbed "information resources" by the W3C Technical Architecture Group.[3] The convention put forward is that URIs for things " outside the Web" should either use a fragment identifier ("#") at the end or a 303 HTTP redirection code. The reason for doing so is that if names for "real world objects" are confused with names for "documents," it could cause problems for Semantic Web inference engines.

The entire point of Linked Data is that a URI *should* have RDF statements available via HTTP *about* the resource(s) denoted by the URI. The convoluted scheme of using either a fragment identifier or HTTP redirection is done in order to create a separate URI for a retrievable representation of a thing that differs from the URI for the thing itself. Thus, a user agent such as a browser can follow one URI to another via the links given by the RDF statements (just as humans follow links), and so can "follow your nose" to discover new RDF statements that form a more complete knowledge representation of the resource. For example, the URI *http://www.example.org/eiffel.rdf#* that stands for the Eiffel Tower itself can automatically resolve in a browser to *http://www.example.org/eiffel.rdf*. Then the information resource *http://www.example.org/eiffel.rdf* about the Eiffel Tower could have a link to *http://www.example.org/France#* so that the RDF retrieved from the latter web-page could lead a reasoner to discover that the Eiffel Tower was in France.

[2] *http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/*
[3] *https://www.w3.org/2001/tag/issues.html*

2 Security Properties of the Semantic Web

So why should Linked Data use HTTPS rather than HTTP URIs? First, it it necessary to state the security goal of encrypting HTTP connections with TLS. Security is defined in terms of an attacker (often called a "threat model"). Informally, if a message is encrypted, an attacker can not discover the original message without a *secret key* that the attacker does not have. The original message is called the "cleartext" and the message encrypted by the secret key is called the "ciphertext." To define this more precisely involves defining the property of *semantic security*, as defined by Goldwasser and Micali: "Whatever is efficiently computable about the cleartext given the ciphertext, is also efficiently computable without the ciphertext" [9]. To rephrase, an attacker can gain nothing in terms of information if the ciphertext has been intercepted. Due to this, it should be safe to send messages in ciphertext across passively monitored or even actively attacked connections. To be succinct, TLS is how HTTP messages are encrypted. Furthermore, these messages are not only encrypted, but the server that delivers the HTTP message can be *authenticated* with a certificate (a public key attached to its domain name), so that the origin website can be proven to have sent the message.[4]

If one takes the links in Linked Data seriously, then an attacker may perform a number of attacks to change how a Semantic Web-enabled application uses data being served as Linked Data on the Web if only HTTP is used. The first attack is trivial: Simply watch for HTTP traffic by a user on a given connection, and since no encryption or authentication is used by HTTP without TLS (i.e. all HTTP messages are sent as plaintext), then the attacker can intercept the HTTP traffic and deliver *whatever data they want* to the unsuspecting user. For example, if one is retrieving open government data in Linked Data about the expenses given by the French government for the upkeep of the Eiffel Tower and the revenue created by tourists visiting the Eiffel Tower, and an attacker wanted to maliciously to prove to the French government that the Eiffel Tower was a bad investment, then the attacker could simply intercept the traffic to the HTTP website and change the numbers in the government data. It would be impossible for a user, such as a journalist, to tell if their HTTP traffic was tampered with by an attacker. This is a very easy attack that can be done using open-source tools such as *wireshark*[5] and *sslstrip*.[6]

The problem only gets worse if the Semantic Web application uses the "follow your nose" algorithm described earlier. If this was indeed the case, then if at *any point* data was retrieved from RDF statements given by a HTTP URI, then the attacker can simply change the data and so influence the Semantic Web inference engine. For example, if a Linked Data-aware application went

[4] In reality, the server proves it has a key validated by a Certificate Authority that the browser accepts. There is a long-standing issue that Certificate Authorities can create certificates for domains they do not own. [1]

[5] *https://www.wireshark.org/*

[6] *https://moxie.org/software/sslstrip/*

to *http://www.example.org/eiffel* and was redirected via a 303 status code to *http://www.example.org/eiffel.rdf*, then that redirection could be changed to a site of the attacker's choosing such as *http://www.example.org/evil.rdf*. If RDF was retrieved from *http://www.example.org/eiffel.rdf* declared that the Eiffel Tower was in Paris, then an attacker could redirect to their malicious *evil.rdf* and state falsehoods such as stating that the Eiffel Tower is actually in a town in the USA rather than a monument in France. This of course would cause errors in the inference engine and any application. So, *any* HTTP URIs that are linked to from RDF can cause problems for a Linked Data application even if the original URI uses HTTPS.

Given that Semantic Web reasoning in Linked Data depends on having trusted information, the *entire* process of reasoning and information retrieval must use TLS for *every* URI if the Semantic Web application is to be trusted. The only exception to this is that if the URI is not accessed, but merely used as an identifier. However, in this case the only trusted Semantic Web inference process is one that does not depend on the HTTP infrastructure. This is a serious problem for the Semantic Web community as the use of TLS-enabled HTTP URIs on the Semantic Web is minuscule, being less than *.1%* of Semantic Web URIs.[7]

3 WebID+TLS considered Harmful

There has been some awareness of TLS in the Semantic Web community due to the *WebID+TLS* effort to use URIs as identifiers for people, with the evocative goal of creating decentralized social networking applications [16]. The goal of WebID is that each person can have their own personal URI that maintains their identity on the Web and from which their personal data, given by RDF statements, can be retrieved. Variations on WebID+TLS have tried adding access control in order make sure that personal data can only be given to explicitly authorized agents rather than given to absolutely anyone who issues a HTTP request [17]. In WebID+TLS, a user generates a client certificate using asymmetric cryptography (which contains a signature given by a private key stored in TLS keystore, as well as a link to their WebID URL) that is stored by the browser.[8] The public key can then be posted to the URI of their WebID URI. When a user wishes to authenticate themselves and authorize the transfer of their own personal RDF data to a third-party, a Semantic Web application can ask the user to authenticate a TLS session using their client certificate to identify their browser, and since the client certificate stored by the browser is signed with the private key that corresponds to the public key on their WebID URI (i.e. the signature on the certificate can be verified using that public key), they can prove that the WebID URI is controlled by the same agent that controls the browser. Normal TLS requires only the server authenticate, but client certificates also allow a client to authenticate, creating a mutually authenticated TLS session.

[7] According to *http://lodlaundromat.org/* (Retrieved on June 30th 2016).
[8] Unfinished specification: *https://www.w3.org/2005/Incubator/webid/spec/tls/*

Despite using TLS, the problem with WebID+TLS is that it violates the security and privacy boundaries of the Web and is based on out-of-date browser cryptography that is being deprecated by the browsers themselves. Today, application security on the Web depends on the *same origin policy*: Any code or data on the user's browser is restricted by origin. The origin is the domain name without the scheme, i.e. *origin.org* without the *http* scheme. So a cookie from *http://origin.org* should be accessible from *https://origin.org* and any subpages such as *http://origin.org/page.html*, but should not be accessible from *http://origin2.org*. The same goes for any information stored in *localStorage* in the browser and any state changes in the browser resulting from Javascript calls. In WebID+TLS, the client certificate is currently created with the `<keygen>` tag and stored in the TLS keystore. However, due to its age (it predates the same origin policy) and the lack of privacy concerns when it was designed, a *keygen*-generated client certificate can be accessible from any origin, and so serve as a "super-cookie" to track users across origins. If an attacker wants to track a user, the attacker can query and ask for a client certificate. A malicious attacker from one origin who wanted to re-identify a user on another origin could simply install a client certificate on the malicious origin and ask for the certificate again on another origin. Worse, the current user-experience around both generating and selecting client certificates is confusing, and neither the installation nor usage of client certificates by HTML is standardized, so the usage of client certificates in HTML depends on ad-hoc browser behavior dependent on a particular idiosyncratic per-browser interpretation of a MIME-type. [9]

Even if the user does end up successfully identifying themselves with a client certificate, current browsers use the insecure MD5 hash function in the signed client certificate. MD5 has proven to not be collision resistant, which means that an attacker can generate a fake client certificate whose signature can be verified using the public key of a user even though the attacker does not have private key of the user [18]. In this way, a user can be impersonated by an attacker. Due to these security and privacy issues, browser vendors are currently deprecating *keygen* from HTML and client certificates handling from the application layer, which will mean *WebID+TLS* will stop working. Although the Semantic Web community has yet to engage wit it, modern cryptographic primitives are now provided by the W3C Web Cryptography API [19]. Getting rid of passwords can be done via authentication by hardware tokens (or other authenticators) by the W3C Web Authentication API, which is designed both to not violate the same-origin policy (i.e. keys differ per origin) and use modern cryptographic primitives such as ECDSA [4]. Much like the rest of the Web, the Semantic Web can use explicit authorization of personal data transfer using IETF standards like OAuth [11]. By separating identities by origin and using modern cryptography, both the privacy and the security of users can be protected while enabling decentralized Semantic Web applications.

[9] *https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/pX5NbX0Xack/kmHsyMGJZAMJ*

4 Fixing Security for the Semantic Web

The Semantic Web was designed without any security considerations. Still, today there is almost no academic work on security in terms of the Semantic Web [14]. Rather unfortunately, there also seems to be considerable confusion about security within the Semantic Web research community, ranging from ignorance of the security problems in HTTP URIs to misuse of TLS in WebID+TLS. This does not have to be the case: A number of simple solutions can be presented to upgrade the Semantic Web to a Secure Semantic Web.

Part of the problem is a confusion over the layers of the Web: TLS is a network-level protocol, not an application level protocol. For example, interrupting a network-level TLS handshake in order to start a user-centric identity and authentication protocol in WebID+TLS is bad design insofar as it mixes the application-level concept of an "a person's identity" with the network level that just ships bits around. To some extent, the problems with the use of TLS on the Semantic Web is that network level information (whether a HTTP connection is encrypted using TLS or not) is exposed on the level of a URI used in a Semantic Web applications, including but not limited to WebID+TLS. URIs are also exposed to HTML links, and thus Tim Berners-Lee is rightfully worried that a switch to HTTP violates his principle that "Cool URIs Don't Change"[10] and so the adoption of HTTPS would break existing links. Berners-lee goes even further:"Put simply, the HTTPS Everywhere campaign taken at face value completely breaks the web. In a way it is arguably a greater threat to the integrity for the web than anything else in its history" [2]. In general, network-level information about encryption (i.e. the use of HTTPS) should not have been exposed to the application level, so two solutions would be that the scheme (HTTP vs. HTTPS) should not matter for applications, or that all HTTP connections can silently upgrade to HTTPS (as put forward in the HTTP 2.0 Working Group [15]. Sadly, it seems too late for these sensible proposals for the Web at large.

Of course, one can still declare by fiat that all HTTPS URIs are equivalent to HTTP URIs on the Semantic Web. The plausible solution Berners-Lee put forward was that "If two URIs differ only in the 's' of 'https:', then they may never be used for different things."[11] However, co-inventor of HTTP Roy Fielding disagreed, noting that "any shared authority between a site on port 80 and a site on 443," where port 80 is typically used for HTTP and port 445 for TLS.[12] Could such a rule work *just* for the Semantic Web? Unforunately, the RDF specification states that HTTP and HTTPS URIs are not the same: "Two IRIs are equal if and only if they are equivalent under Simple String Comparison ... further normalization MUST NOT be performed when comparing IRIs for equality." [8]. Using *owl:sameAs* (or to preserve the correct semantics, *owl:equivalentClass* and *owl:equivalentProperty* when needed) between HTTP and HTTPS URIs does not work as these are statements about the things a URI denotes, not the text of

[10] *https://www.w3.org/Provider/Style/URI.html*
[11] *https://lists.w3.org/Archives/Public/semantic-web/2014Aug/0078.html*
[12] *https://lists.w3.org/Archives/Public/semantic-web/2014Aug/0089.html*

the URI itself. RDF also does not have a way to talk about a URI itself via a mechanism such as quoting. Even if one attempted to hack HTTP and HTTPS equivalence together, it would require identity statements for every property, class, and instance. While this might patch together a reasoner to do the same inferences over HTTPS that it would over HTTP URIs, it would still allow the application to be attacked if it followed Linked Data principles and tried to retrieve data from any HTTP URI.

Another solution noted by Berners-Lee is:"'The HTTP protocol can and by default is upgraded to use TLS without having to use a different URI prefix" [2]. Although newer versions of HTTP 2.0 does not require HTTPS (although opportunistic encryption is still under discussion [15]), opportunistic encryption that automatically upgrades HTTP to HTTPS can be done by using the W3C Upgrade Insecure Requests specification where a server requests that a HTTPS URI be used if possible via a HTTP Header [20]. The server can then use a HSTS header to prevent any downgrade attacks that stripped the 'S' off the HTTPS in a URI [13]. In this way, a browser or Semantic Web application can ask for a Semantic Web HTTP URI and retrieve content from a HTTPS URI. The W3C began to use this methodology on its site, including RDF namespaces.[13] Yet when the W3C asked the Linked Data community if they would want to consider HTTPS URIs to be equivalent to HTTP URIs,[14] this was not accepted due to the fact that the W3C specifications do not allow anything except exact matching of strings for equality as mentioned earlier [8], while other W3C staff member Sandro Hawke put his hope in the eventual use of Upgrade Insecure Requests and HSTS, and to just "keep writing 'http:' and trust that the infrastructure will quietly switch over to TLS."[15] Nonetheless, this path makes insecure HTTP URIs the default mode, and does not offer any reason for the Semantic Web to upgrade to TLS. Worse, as the Upgrade Insecure Requests headers are delivered over HTTP, any attacker actively watching the unencrypted HTTP redirection can simply strip those headers to prevent upgrade to HTTPS and allow the HTTP content to be attacked. So to just keep using HTTP URIs and hope for the best does not solve the problem.

Shouldn't W3C Recommendations, rather than being considered as religious texts, be fixed to keep up with modern security? Although the RDF specification lack a way to discuss URIs themselves [8], it would make sense that best practices allow HTTP and HTTPS URIs to be equivalent. While this could be added to future editions of the specification, there is no reason to wait. Furthermore, a preference should be given to HTTPS URIs to encourage the eventual deprecation of HTTP URIs, just as is done on the ordinary Web. Therefore, new Semantic Web URIs *should* use TLS and HTTPS URIs. Older URIs may upgrade using Upgrade Insecure Requests [20], but it should not be encouraged to keep writing HTTP and hope for the best on the modern Web. Eventually, all

[13] *https://www.w3.org/blog/2016/05/https-and-the-semantic-weblinked-data/*
[14] *https://lists.w3.org/Archives/Public/semantic-web/2016May/0082.html*
[15] *https://www.w3.org/blog/2016/05/https-and-the-semantic-weblinked-data/#comment-93683*

HTTP URIs on the Semantic Web should be deprecated. If one believes that the use of URIs on the Semantic Web is still a marginal phenomena, the argument that this "breaks" existing Semantic Web software seems to assume that the Semantic Web is a mature technology with widespread adoption. The Semantic Web still is in its early stages, and so doing security right should take precedence over preserving broken software.

The future of HTTPS is also bright: TLS has improved and became easier to deploy as well, with certificates available for free and the protocol itself faster and more secure. For example, TLS version 1.3[16] corrects a number of problems in TLS 1.2 such as an unclear state machine [3] and attacks on TLS authentication [6]. These problems and more are fixed in TLS 1.3, and there is now provably secure implementations of TLS 1.3 that rely on fast elliptic curve cryptography [5]. Earlier, the price of server-side certificates was considered too high and there was concerns over the hierarchical nature of the Certificate Authority system, as a Certificate Authority can issue a certificate that has global scope. These problems led to either Semantic Web researchers completely ignoring TLS due to supposed "security concerns" and for those Semantic technologies using TLS to such as WebID+TLS to want to use "self-signed" certificates instead of those from a certificate authority. These concerns with TLS are now effectively addressed. Thanks to the effort by "Let's Encrypt," the price of server certificates is now free, so there is no reason not to use the high-security TLS certificates from "Let's Encrypt" other than the time investment to configure a Web server to use TLS.[17] Also, if one uses certificates and is worried about a rogue Certificate Authority issuing certificates incorrectly or being compromised by a malicious actor, the IETF Certificate Transparency standards, employed by Google, provides efficient auditing and search of certificates using a Merkle tree.[18]

Some security is always better than none, and there can be no worse security than absolutely no security. Thus, there is no excuse to use HTTP URIs unless they are not dereferenced for further RDF or even HTML data. If this is the case, then these URIs should result in HTTP 200 status error codes. Having redirection via 303 from a HTTP URI and even Upgrade Insecure Requests is possibly dangerous as an attacker could intercept the redirect as the original URI was not using HTTPS (and so redirection and headers happen using cleartext over HTTP). Redirection in Semantic Web URIs should be discouraged in favor of just using HTTPS URIs. Rather than use WebID+TLS, Semantic Web applications should respect the same-origin policy and use modern methods of authentication, such as the Web Authentication API, and authorization, such as OAuth, that respect the same origin policy. If cryptographic primitives are to be used on new protocols on the Semantic Web, they should use the primitives provided by modern APIs and not broken cryptographic primitives whose interaction with the browser are not normatively specified. Given that TLS cer-

[16] *https://github.com/tlswg/tls13-spec*
[17] *https://letsencrypt.org/*
[18] *https://www.certificate-transparency.org/*

tificates are free and that the Semantic Web has still not reached widespread usage enough to argue that moving from HTTP to HTTPS would break real-world applications, Semantic Web tools and Semantic Web vocabularies should switch as soon as possible to using TLS-encrypted HTTPS URIs. There is no excuse not to use encryption if you want users to trust the Semantic Web. Both open data and personal data require security, and personal data in addition also requires privacy, and the building block for both is proper usage of TLS. Otherwise, the use of the Semantic Web will likely be replaced by technology, such as blockchains, that takes security on board in its design.

5 Acknowledgments

This work is funded in part by the European Commission H2020 European Commission through the NEXTLEAP Project (Grant No. 6882).

References

1. Hadi Asghari, Michel Van Eeten, Axel Arnbak, and Nico Van Eijk. Security economics in the HTTPS value chain. In *Twelfth Workshop on the Economics of Information Security (WEIS 2013), Washington, DC*, 2013.
2. Tim Berners-Lee. Web security - "HTTPS Everywhere" harmful, 2015. https://www.w3.org/DesignIssues/Security-NotTheS.html.
3. Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE, 2015.
4. Vijay Bharadwaj, Hubert Le Van Gong, Dirk Balfanz, Alexei Czeskis, Arnar Birgisson, Jeff Hodges, Michael Jones, Rolf Lindemann, and J.C. Jones. Web Authentication: An API for accessing scoped credentials, 2016. https://www.w3.org/TR/webauthn/.
5. Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. Proving the TLS handshake secure (as it is). In *International Cryptology Conference*, pages 235–255. Springer, 2014.
6. Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113. IEEE, 2014.
7. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.
8. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax, 2014. https://www.w3.org/TR/rdf11-concepts/.
9. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.
10. Harry Halpin. Social semantics. In *Social Semantics*, pages 187–203. Springer, 2013.

11. Dickt Hardt. The Oauth 2.0 authorization framework, 2012. https://tools.ietf.org/html/rfc6749.
12. Patrick J Hayes and Harry Halpin. In defense of ambiguity. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 4(2):1–18, 2008.
13. J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (hsts), 2012. https://tools.ietf.org/html/rfc6797.
14. Adis Medić and Adis Golubović. Making secure semantic web. *Universal Journal of Computer Science and Engineering Technology*, 1(2):99–104, 2010.
15. M. Nottingham and M. Thomson. Opportunistic security for HTTP, 2016. https://www.ietf.org/id/draft-ietf-httpbis-http2-encryption-07.txt.
16. Henry Story, Bruno Harbulot, Ian Jacobi, and Mike Jones. FOAF+SSL: Restful authentication for the social web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*, 2009.
17. Sebastian Tramp, Henry Story, Andrei Sambra, Philipp Frischmuth, Michael Martin, and Sören Auer. Extending the WebID protocol with access delegation. In *Proceedings of the Third International Conference on Consuming Linked Data-Volume 905*, pages 99–111. CEUR-WS. org, 2012.
18. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 19–35. Springer, 2005.
19. Mark Watson. Web Cryptography API, 2015. http://www.w3.org/TR/WebCryptoAPI/.
20. Mike West. Upgrade Insecure Requests, 2015. https://www.w3.org/TR/upgrade-insecure-requests/.