W3Bcrypt: Encryption as a Stylesheet

Angelos Stavrou, Michael E. Locasto, and Angelos D. Keromytis

Department of Computer Science
Columbia University
1214 Amsterdam Avenue
Mailcode 0401
New York, NY 10027
{*angel, locasto, angelos*}*@cs.columbia.edu*

Abstract. While web-based communications (*e.g.,* webmail or web chatrooms) are increasingly protected by transport-layer cryptographic mechanisms, such as the SSL/TLS protocol, there are many situations where even the web server (or its operator) cannot be trusted. The end-to-end (E2E) encryption of data becomes increasingly important in these trust models to protect the confidentiality and integrity of the data against snooping and modification.
We introduce W3Bcrypt, an extension to the Mozilla Firefox platform that enables application-level cryptographic protection for web content. In effect, we view cryptographic operations as a type of style to be applied to web content, similar to and along with layout and coloring operations. Among the main benefits of using encryption as a stylesheet are (*a*) reduced workload on the web server, (*b*) targeted content publication, and (*c*) greatly increased privacy. This paper discusses our implementation for Firefox, although the core ideas are applicable to most current browsers.

Keywords: E2E cryptography, web security, cryptographic applications

1 Introduction

The growth in popularity of hosted web services (including online merchants, blogging, and webmail) offers new possibilities for commerce and communication. Unfortunately, most of these services are hosted by third parties that should not be trusted with the content of the messages that are passed between content publisher and the reader. As a simple example, users of popular webmail services like MSN Hotmail or Google's Gmail must trust that MSN or Google will respect the confidentiality of their mail messages. Of course, the user could employ PGP or S/MIME for email messages, but this presupposes that the webmail service can be accessed by a trusted mail client. The webmail interfaces of these services do not provide such a trustworthy client. Even if these interfaces supported client-side PGP operations (via ActiveX, Flash, or a Java applet), users cannot trust these components with their private key or passphrase.

Our goal is to build a trustworthy client-side environment into web browsers that is independent of the service provider. This environment need not be limited to webmail services, but it should support treating any web-service provider as a transit conduit for an opaque block of encrypted and integrity–protected data.

1.1 E2E Security For Web Content

Traditional methods for confidentiality and integrity involve the use of cryptography in the middle of a communications pathway. Communications involving web content can be protected at several layers in the network stack. Connections between client and server could be secured at the network level using IPsec. General-purpose web servers typically employ transport layer encryption.

To the casual observer, these techniques may seem identical. Each can protect the confidentiality and integrity of the content in transit. In reality, these approaches are orthogonal to each other and have fundamental differences. They operate at different levels in the network and protect different notions of "content." Most importantly, neither can protect from a compromised or malicious application server because their confidentiality and integrity[1] protections do not reach up through the application layer.

The privacy and security of web content has usually been addressed by TLS/SSL. Encryption at this layer presumes that the application provider is trustworthy, just as encryption at the network level (*e.g.,* IPsec) assumes that the endpoints are trustworthy. In a growing number of scenarios, it is undesirable, if not unreasonable, for users to trust the communications provider with the confidentiality and integrity of their data. For example, a blogger (Bob) may not trust his hosting provider, or a customer (Alice) may not trust a commercial webmail service with her banking information. Currently, the blogger is forced to trust the blog hosting service and has no expectation of confidentiality between himself and his readers. Likewise, a customer purchasing items from an online store has to divulge sensitive personal and financial information to the merchant. Revealing such information to an online store is an unacceptable risk, especially since such entities cannot guarantee the security of their systems against electronic (or physical) theft leading to identity theft.

For situations where we cannot trust the service provider, we advocate the use of end-to-end (E2E) encryption where the endpoints are the actual users (or as close to them as possible). Not only is E2E encryption good for the privacy and security of the end user, but it is unexpectedly beneficial for service providers as well. A recent example is AOL's decision to allow users of its AIM instant messaging service to encrypt their conversations E2E. The alternative would have been for AOL to set up SSL connections for each conversation taking place on their network. Not only does the latter choice insert AOL's servers as *de facto* men-in-the-middle (and thus violate the users' expectation of privacy for an encrypted conversation), it places an unreasonable performance demand on AOL's servers. Assuming that AOL's business model does not require examining

[1] We specifically choose not to address availability in this paper, as it would be trivial for the service provider to impose a DoS on the user. Such an occurrence is anathema to the concept of being a useful provider of services.

AIM traffic, the use of E2E cryptography avoids performance issues and the associated cost of hardware, systems, and management.

1.2 Contributions

A better system would allow both the blogger and customer to treat the service providers as a mere communications pipe through which they can tunnel confidential information to their target audience (in Bob's case, his readers or subscribers; in Alice's, her financial institution).

We present W3Bcrypt, a system for transparent E2E encryption of web content that uses public-key cryptography (*e.g.,* PGP). W3Bcrypt can be thought of as another layer of HTML rendering; in effect, we treat encryption as another style applied to content. W3Bcrypt makes three major contributions:

- **An E2E privacy-enhancing browser extension:** W3Bcrypt provides confidentiality and integrity to content producers who wish to publish to a set of readers. The system also supports the ability for customers of web merchants to communicate with their financial institution, and a way for users of webmail systems to employ PGP even if the interface does not support it.
- **The offload of cryptographic processing** from the server to clients. SSL has typically been used to protect web communications. However, SSL places a burden on servers that only increases with the number of clients. With W3Bcrypt, the burden of cryptographic operations is placed mostly on the client – content only has to be encrypted once in the server data store. While W3Bcrypt is not meant to replace SSL, it can complement SSL to provide a net gain in security (defense in depth) against multiple threats.
- **The concept of cryptographic processing as another phase of styling web content:** Just as content is rendered by the browser for placement, size, and coloring, so too can the content be decoded into something the user is authorized to view.

The remainder of this paper discusses the design and implementation of W3Bcrypt as well as background work on SSL, web spoofing attacks, and browser security. We also provide a security analysis of the system and present a performance evaluation.

2 Approach

The W3Bcrypt package is an extension to Firefox that permits a publisher to securely convey content to a consumer at the application level in an end-to-end fashion. The core functionality is the ability to perform PGP (or similar) cryptographic operations on blocks of web content. To support these features, the extension includes changes affecting layout operations, small additions to the UI and the ability to invoke PGP. Since one major goal is to refrain from modifying the source code of the browser, the new features are packaged as an extension for easy installation, upgrade, and removal. Adding encryption as a style takes advantage of the power of CSS, because no new tags need to be added to the HTML grammar. Work is being done in that vein on XML encryption [9].

This section discusses our primary use cases, presents a security analysis of the system (including the threat model, attacks, and potential countermeasures), and talks about some of our limitations. Section 3 discusses the actual implementation of the Firefox extension. We provide an analysis of the system's performance in Section 4.

2.1 Use Cases

We are motivated to build and analyze W3Bcrypt to enhance the amount of privacy provided by the current web infrastructure. Privacy, in this case, refers to the confidentiality and integrity of web content – the system is not used to obfuscate referrer headers or similar information, although it could be leveraged for this purpose. Below, we offer three situations we have personally encountered as motivating examples and use cases.

1. Web *peers* may wish to exchange information through a public email service such as Google's Gmail or Microsoft's Hotmail. Currently, these services are at liberty to scan, data mine, and store the content of the peers' messages. While PGP has traditionally been used to protect email communications, the use of a webmail client makes it difficult to use PGP because the browser has no built-in application-level key handling, and any code (ActiveX, Java, Javascript) from the webmail provider cannot be trusted to not divulge the private key or user passphrase. W3Bcrypt solves this problem by providing such a trustworthy client-side environment.
2. A *customer* of an online merchant may wish to use the merchant as a transit network or information conduit by passing an opaque block of data (encompassing the customer's account number and billing address) to the customer's financial institution via the merchant. The bank or credit card company then authorizes payment to the merchant without the merchant knowing the customer's account number(s). In addition, the customer could encode her shipping address such that the merchant does not know where items are shipped, but the transportation agent (*e.g.,* the USPS, UPS, or FedEx) can decode the address and deliver goods as appropriate. We discuss a possible attack on this protection scheme in Section 2.3.
3. A web content *publisher* may wish to forgo *or* supplement traditional authentication and authorization services by publishing content under a specific "audience" key (or series of such keys). Publishers can include bloggers and other content producers like news organizations or media companies.

2.2 Security Analysis

We present a security analysis for our major use cases, including the threat model for each, attacks on the system, and countermeasures that the system provides or could provide with additional implementation or support from the browser.

Threat Model In all of our proposed use cases, our threat model is based on the concept of an untrustworthy service provider as the attacker. In the *publisher* use case, the attacker is the blog-hosting provider or media-content-hosting network. In the *customer* use case, the attacker is the online merchant. In the *peer* use case, the attacker is the webmail provider. The service provider could be merely curious, compromised, or actively malicious. We are not primarily interested in defending against passive or active attackers who attempt to sniff or otherwise control the communications links between the service provider and the client. These traditional attacks can be addressed by using SSL, but SSL cannot protect against a service provider because the provider controls the application level and has access to the data after it has been processed by SSL.

Except for a special case (the Hitchhiker attack), we do not consider any defense against client-side attacks such as trojan horses, viruses, spyware, or other malware on the user's host. In all cases, the attacker is interested in violating the confidentiality and/or integrity of the user's content. We believe that for most situations the availability of content is not an issue; a service provider that denies service is not a very effective service provider, and it is trivial to cause DoS by changing the server to interrupt connections containing PGP content.

Yet, it is a very real possibility that the service provider has defined the ability to examine user content as a core competency or central business need. This type of service provider will therefore be satisfied with imposing a denial of service on users that violate an agreement stating that users are not allowed to obfuscate, encrypt, or otherwise hide their content from the service provider. We consider the existence of W3Bcrypt problematic for such service providers and demonstrate an attack that they could carry out to get around the protections afforded by W3Bcrypt (the aforementioned special case).

We also exclude attacks on the content after it has left W3Bcrypt's purview. For example, a news provider may wish to employ W3Bcrypt as part of a type of DRM scheme where content is targeted to a specific consumer or group of consumers. After the consumer's W3Bcrypt system has decrypted the content, the consumer is free to copy the content and pass it on. Since the content is out of W3Bcrypt's control at this point, we do not consider this part of our threat model. We note that this type of attack exists for all DRM or content distribution schemes. Furthermore, the threat model in these situations is different from ours – we assume that the receiver is free to do whatever they want with content directed at them.

Attacks There are two major types of attacks against W3Bcrypt: the brute force attack and the Hitchhiker attack. We discuss the possibility of replay attacks in section 2.3. The brute force attack is carried out by a service provider that attempts to discover the private key being used to sign or decrypt content. We assume that W3Bcrypt is no stronger against this attack than PGP itself. The attacker could also attempt to gain the key through coercion or economic incentives, an attack that is effective against any cryptographic scheme.

The Hitchhiker attack is very interesting in that the attacker does not try to directly subvert or control the cryptosystem. Instead, the service provider attempts to piggyback code that bypasses the cryptographic controls on the content and accesses the content either before or after it has passed through the cryptosystem. Additionally, the attacker can attempt to insert web content that is meant to masquerade or blend in with the encoded or decoded web content. This attack is a type of spoofing attack. If the attacker has not discovered the private key, he or she cannot forge a signature for such content.

For example, a webmail provider may include, as part of their webmail client, Javascript that captures a user's keystrokes. If the user later uses W3Bcrypt to encode the mail message, the webmail provider still has a log of the plaintext by virtue of the keystroke monitoring. Such monitoring is already done to support automatic spell-check and automatic saving functionality in some webmail applications. On the receiving side, the attacker could include Javascript that attempts to read the contents of a message once it has been decrypted by W3Bcrypt.

Countermeasures In order to overcome the Hitchhiker attack, the browser would ideally support a policy-driven mandatory access control on a fine-grained namespace framework for the browser objects, like SELinux does for the Linux operating system. Lacking such controls, we can attempt to perform input operations in a transparent overlay frame, encrypt the content in this overlay frame, and then transfer it to the target element in encrypted form. Likewise, encrypted content can be transferred to a new overlay frame and decrypted in that context with "external" Javascript disabled or unable to read or write to that frame.

This solution still leaves open the question of a user that unknowingly includes malicious Javascript in the content they have encrypted. The solution to this problem is an open area of research. One potential (but unappealing) solution is to employ some form of model or proof-carrying code [16] combined with a policy mechanism like the Java Policy and Permissions framework.

2.3 Limitations & Discussion

W3Bcrypt currently depends on the presence of the GnuPG software package and invokes an *xterm* to call the *gpg* tool. We plan to improve W3Bcrypt so that it can detect and use other PGP packages. We are also investigating the use of the appropriate command shell tool so that W3Bcrypt can be used on the Windows version of Firefox.

We note that W3Bcrypt alone does not support the *customer* use case. Merchants and financial institutions would need to modify their systems to expect

PGP encoded data and process it properly. In particular, the online merchant would need to alter input validation routines for the protected data.

Key Management As with all systems that employ a form of public key cryptography, the issue of key management is important. We refrain from dealing with key management or revocation. W3Bcrypt's design avoids the use of a large scale PKI and employs the peer-to-peer "web of trust" approach implicit with user-managed PGP keys. Key creation, sharing, signing, and revocation are explicitly not handled by the current tool. Instead, these operations are deferred to the underlying PGP package. We plan to include GUI entry points (*e.g.,* a "buddy" list) to this functionality in later releases.

Privacy Preferences While our goal is to enhance privacy, W3Bcrypt does not take advantage of or interface with the Platform for Privacy Preferences (P3P) initiative [6], nor is it meant to directly support other browser-based privacy-enhancing mechanisms like referrer header rewriting or blanking, although it could be used to do so. Determining if the use of W3Bcrypt can benefit these areas is an open problem.

Integration Complexity In the customer use case, the customer needs to communicate with at least three entities: the merchant, to select goods and create the order; her financial institution, to arrange payment of the final sum; and a shipping agent, to arrange a particular type of shipping. The merchant, in order to make online shopping attractive to the customer, must integrate the latter two communications into its online shopping process. Since merchants currently expect to at least parse the customer information for sanity, a customer using W3Bcrypt would require the merchant to partially rewrite their web application and modify their database.

While the system provides the basis for a number of use cases, Schneier reminds us that security is a process, not a product. In many use cases, W3Bcrypt handily fills an immediate need. In other situations, such as those involving complex, multi-party protocols, W3Bcrypt alone does not provide adequate privacy against higher-level attacks. As a simple example, the merchant use case assumes that the customer wants to hide both her financial credentials and shipping address from the merchant. However, the shipping agent usually prices service according to location and delivery method. If the shipping agent returns this information in a plaintext format to the merchant, the merchant could potentially guess the location of the customer (especially if the information is correlated with information gleaned from IP address geo-location services).

In theory, these problems are not difficult to solve. The customer should merely set up a key pair with the chosen shipping agent, financial institution, *etc.* In practice, this key management may prove difficult, and leaves open the large question of how this sort of information integration actually occurs in the merchant's web application.

Furthermore, any of the use cases could suffer from replay attacks, although such an attack would be more noticeable and presumably not as harmful in the *peer* and *publisher* use cases. Duplicate blog postings or emails will probably be recognized as such and ignored (even if their content were relatively dire –

for example, an inflammatory news bulletin or letter of dismissal). More care must be taken in the *customer* use case. The customer should include some randomness in the data to be communicated to their financial institution. A timestamp, sequence number, or randomly generated ticket prepended to the account number would serve to identify duplicate transactions submitted by the merchant. W3Bcrypt does not currently support transparently concatenating a timestamp to all encrypted fields, but this capability is straightforward to add.

Even with these limitations, we believe our system is immediately useful, and we employ it almost daily. We look forward to incorporating some of the countermeasures and solutions to the limitations in our ongoing development of the Firefox extension, which we describe next.

3 Implementation

We have implemented W3Bcrypt as an extension to version 1.5.x of the Mozilla Firefox web browser, although the core ideas are browser-agnostic. W3Bcrypt could easily be implemented in other popular browsers like Opera, Safari, and Microsoft Internet Explorer, or even a text-based browser like *wget* or *lynx*. Our extension is available at our website[2]. The major features of the W3Bcrypt system include the ability to transform chunks of HTML content from and to PGP-encrypted, ASCII-armored blocks of data. In addition, the system supports the ability for the browser to automatically decrypt `div`'s marked with a special CSS class id. The most immediately useful feature is the ability to select free text in form objects like textareas and textboxes, access the context menu, and utilize one of the basic PGP functions from a menu of six: encrypt, sign, encrypt and sign, decrypt, verify, and decrypt and verify.

3.1 Package Layout

Our prototype adheres to the packaging conventions for Firefox 1.5.x extensions. The system is comprised of four files: *install.rdf*, *chrome.manifest*, *overlay.js*, and *overlay.xul*. The first two files are used during installation of the extension and contain the metadata that describe the package and its capabilities. In particular, the *chrome.manifest* file contains directives that overlay our new widgets on the standard browser GUI components.

The latter two files contain the bulk of our implementation, and they are located in the *chrome/content/* subdirectory of the extension XPI file. They reflect a clean split between the new GUI components and the raw functionality for invoking GPG. The XUL file defines a new sub-menu for the context menu The JS file contains Javascript functions that invoke the GPG functionality via an *xterm*, and it supplies a function to automatically decrypt marked `div`'s.

[2] `http://nsl.cs.columbia.edu/projects/w3bcypt/`

3.2 Integration with GPG

One of the design goals of W3Bcrypt was to provide a quick manual method for invoking the extension functionality. To simplify implementation, we made a design decision to leverage any PGP software already installed on the host. We currently use GPG, which is available for both Windows and Linux (and a number of other platforms). We decided to implement the six major cryptographic operations as choices in the context menu. These choices are gathered into a submenu to avoid crowding the regular context menu. The various functions in *overlay.js* are accessible via this context menu. These functions do some setup work (gathering content, setting up temporary files, creating an *xterm*) and then delegate to *gpg*. The result is gathered and written into the HTML element it originated from, via the `innerHTML` attribute.

Fig. 1. *Encrypted Data.* After the text is selected and W3Bcrypt is invoked via the context menu, the selected text is replaced with ASCII-armored data. This data can be decrypted by the receiver either manually or automatically.

3.3 Auto-rendering of encrypted DIVs

One of our primary goals is to treat cryptographic content as another type of style. To this end, the prototype recognizes specially marked `div` elements (those with the class attribute set to "w3bcrypt" as follows) and automatically decrypts them.

```
<div class="w3bcrypt">
 ...encrypted content here...
</div>
```

When Firefox finishes loading the DOM for a page, the extension requests a list of all `div` elements marked with the *w3bcrypt* class and proceeds to decrypt them, prompting the user for his passphrase. Only the decrypt and verify operations are automated for marked `div`'s, as this arrangement alleviates the burden of manually selecting some text and decrypting it via the context menu.

4 System Evaluation

In order to make sure that the cost of employing W3Bcrypt is justifiable, we need to quantify the impact of the system on the resources (*i.e.,* space and time) used by both the client and server. We employed two metrics to evaluate the performance of the system. For the client, we focus on the overhead due to encryption and decryption operations. On the server, we are most concerned with the difference in storage requirements for the encrypted and plain content. We classify objects as text or binary to differentiate between two cases: the content stored in a database, which is mostly text, and the content served by a web server (likely a mixture of both text and binary objects). We ignore the initial cost of the encryption of the server's web or database content since it happens only once and it is not repeated for any of the subsequent client requests.

For these experiments, we used a machine with a 2.7 GHz Intel Pentium 4 processor with 1GB of RAM running a Debian Linux distribution. Cryptographic operations were provided by GPG version 1.4.2 with the ASCII armored output option enabled. All of the results presented are the computed averages of multiple experimental runs with tight confidence intervals.

We used two different types of datasets in our experiment: a text repository containing the American Constitution[3] and three commercial web sites[4]. For the web sites, we stored and used all the data returned when accessing their first page, including the index page and any other pop-up, overlay, or roll-over objects that appeared as a result of scripting. This type of capture results in slightly larger web content sizes than what we usually expect.

4.1 Encrypted versus Plaintext Content Size

The pure text experiments use parts of the plaintext version of the American Constitution. Figure 3 shows that for small text sizes there is a significant increase in the space required for the generated ciphertext. For text sizes above a threshold (about 2KB), we observe the opposite effect: a significant drop in the space requirements. This is because GPG uses GNU ZLIB compression library to compress, aiming to effectively increase the entropy of the files, before

[3] `http://www.house.gov/Constitution/Constitution.html`
[4] `http://www.cnn.com`, `www.nytimes.com` and `www.chase.com`.

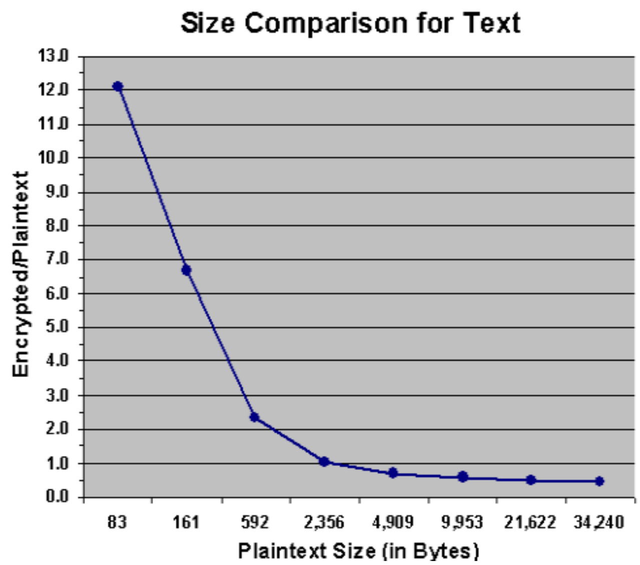

Fig. 2. *Comparison of Size for Encrypted vs Plaintext.* Small plaintext size have a significant increase in the generated encrypted text size. This size drops sharply to values below 1 for plaintext sizes of more than 2KB.

encrypting them. For smaller text sizes, this compression does not work very well, and we can observe the opposite effect – the produced ciphertext file increases in size. Of course, GPG supports other, more sophisticated, compression algorithms which can possibly improve the encrypted file sizes for all file types.

We conducted experiments involving real web content. Figure 4 depicts our results. In general, binary objects like images (high entropy) demonstrate size inflation whereas pure textual objects (*e.g.,* HTML and Javascript) undergo a size decrease. However, the images are usually of bigger cumulative size and the overall result is a rise in storage requirements for the encrypted files. This rise is proportional to the initial content size, and it is almost always no more than twice the size of the original unencrypted web content.

4.2 Overhead of Encryption & Decryption Operations

W3Bcrypt employs two types of actions: manual operations requiring user intervention (encryption), and the decryption operation, which happens automatically when the browser detects an encrypted object. We measure the latency overhead from the automatic execution of an encryption or decryption of an object or set of objects. This penalty is what really matters to the end user. When downloading an encrypted page, the user wants to see how much longer it will take for the page to complete. This type of measurement avoids any comparison

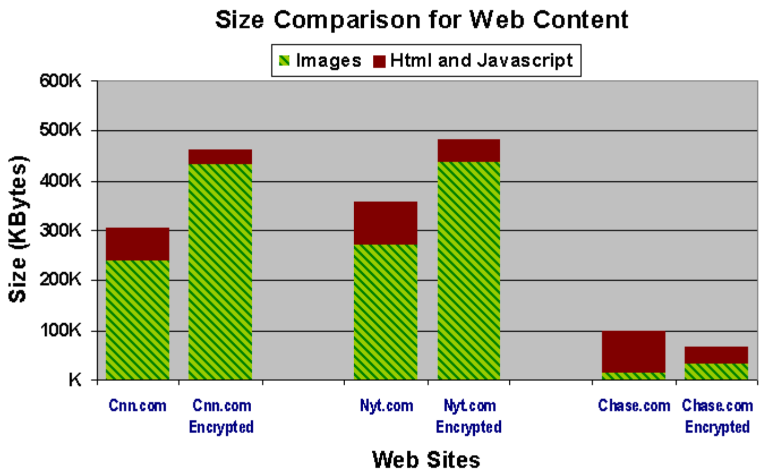

Fig. 3. *Web Content Size Comparison.* There is an increase in size for the produced encrypted text which is proportional to the initial size of the site. This increase comes solely from the encryption of the binary images (lower portion of the bars). HTML and Javascript files decrease in size (upper portion of the bars).

between a manual and an automated operation and thus avoids the need to take user-browser interaction into account.

Figure 4 displays the results of both operations on different web sites. Decryption takes less time than encryption both for HTML and for images. For sites that offer content of smaller size, such as the `chase.com` website, the decryption process requires just over half second to complete. We note that the latency overhead depends both on the size of the site and the number of objects that it contains. This dependency is an artifact of our implementation. We use a different call to GPG for cryptographic operations on each object. This organization generates extra overhead for the system since a page might contain multiple objects. The impact is clear if we compare the encryption and decryption time for the first two sites, in Figure 5. Although both sites have similar content size, the first (`cnn.com`) has many more objects than the second (`nyt.com`) (see Table 4.2) resulting in almost double the amount of time required.

Table 1. Web sites: size and number of objects

Site name	Total Size(Bytes)	# objects
cnn.com	305,586	137
nyt.com	357,378	88
chase.com	99,504	22

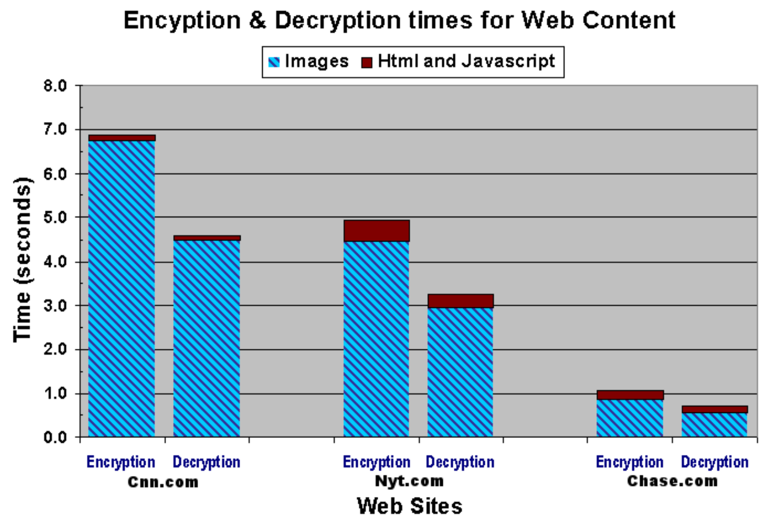

Fig. 4. *Encryption and Decryption Overhead.* The bars display the cumulative time for different operations and for different site content. The lower portion of the bar indicates the time required for the binary and the upper portion for the textual content. The latency overhead depends on the size of the site and the number of object that it contains.

On the other hand, all pure text operations are almost instantaneous, only taking a few milliseconds even for large chunks of text, as shown in Figure 5. Our tool excels when it is used for textual objects. Performance drops when the tool is used on binary objects, but the observed times can be improved by an implementation that first loads the page and then operates on all the encrypted objects with just one call to the necessary library function.

5 Related Work

Our work on E2E confidentiality and integrity protection for web content draws naturally on a number of related efforts in cryptography and web engineering. In particular, work on XML encryption faces many of the same technical challenges. Recent work on encrypting RSS feeds provides extra motivation, while the performance and security analysis of SSL provides some insight into how W3Bcrypt can enhance security while decreasing (or at least not significantly adding to) the performance burden for servers. Finally, work on trusted paths for browsers and more secure browser architectures is of interest because W3Bcrypt can provide some level of visual disambiguation. Since non-decodable PGP blocks are rendered poorly (or not at all) by the extension, they provide visual cues that the content was not meant for the viewer (or represents untrustable content most probably injected by a phisher). The work on more secure browser architectures

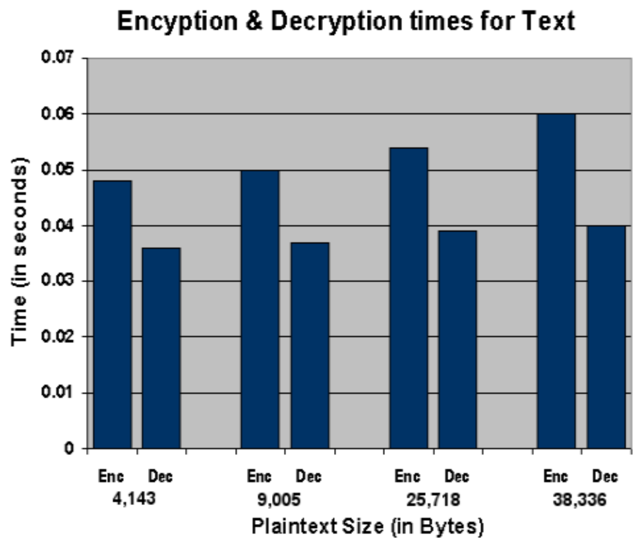

Fig. 5. *Latency overhead for text.* For text objects the latency overhead is just few milliseconds. Such a delay is unnoticeable to the end user.

is of use for cases where the service provider attempts the Hitchhiker attack by including Javascript code that tries to discover the encrypted content. During our research, we were alerted to a parallel suggestion by Gregorio [7] to use GreaseMonkey for encrypting RSS feeds. We take this as an encouraging sign that the problem we are working on is a current and meaningful one.

XML Encryption Some work has been done on content encryption using XSLT and XML. Work suggesting the element-level encryption of XML content appeared as early as April of 2000 [13]. This work, and efforts related to it [9, 10], are complementary to W3Bcrypt. W3Bcrypt currently treats the contents of a `div` element as a single-level block of content. The results are undefined if the content includes HTML markup, although our tests show that Firefox *does* successfully render the HTML markup in the auto-decrypted content. In addition, the goals of the XML encryption projects are quite similar to some of our use cases, especially the *customer* scenario.

SSL Encryption SSL is widely used to secure transport layer communication, by providing confidentiality and integrity for sessions between a web server and a web client. However, SSL is not immune to attacks [5, 2], and since it operates at the transport layer its use assumes at least a trusted server application. We do not argue for replacing SSL; rather, we advocate for augmenting security at another layer.

The use of SSL imposes a hefty performance penalty on servers, and much work has been done to decrease this performance hit. Coarfa *et al.* [4] provide an analysis of the bottlenecks for SSL processing and propose some adjustments to alleviate them. Various other mechanisms for speeding up SSL by both distributing the work [12, 14] and speeding up the underlying cryptographic operations [8] have been proposed.

Other Work Phishing is an attack that has grown in popularity. Both the Spoofguard [3] system and Ye and Smith [17] discuss various methods of creating a trusted path from the server to the user. Both of these systems extend the browser to accomplish client-side protection. While W3Bcrypt is not explicitly built to counter phishing or spoofing attacks, it could be leveraged to display trusted content by decrypting the entire page. Injected content would not decrypt properly (assuming that the attacker does not know the encryption key).

Ross *et al.* [15] implement a browser extension to generate passwords on a per-website basis. This work is complimentary in that it explores ways to protect multiple secrets against malicious websites. It also transparently addresses the tendency to use the same password across multiple sites.

One of the more interesting attacks against W3Bcrypt is the Hitchhiker attack. This attack is a type of cross-site scripting attack, enabled by the ability of the attacker to piggyback Javascript code onto the page. If the browser does not provide namespace separation and access controls (as suggested by Anupam and Mayer [1]), then this Javascript can read content that is meant to be protected by our system. There has been some work on providing a secure browsing environment [11] using sub-process sandboxing and privilege separation. Finally, trusted path techniques (such as randomizing elements of the extension's dialog components) can help in the case of Hitchhiker code that attempts to steal the user's passphrase by displaying a fake dialog. In addition, we can store the user's passphrase so they only have to enter it once per session (identical approaches are taken by *ssh-agent* and desktop mail clients).

6 Conclusions

The growth of hosted web services introduces new methods of communication, collaboration, and commerce. In many of these situations, the client cannot trust the service provider with the confidentiality and integrity of the client's data. W3Bcrypt is a practical and effective mechanism that supports the E2E confidentiality and integrity of web content. Our implementation is an extension to the Firefox web platform and supplies a trustworthy client-side environment for performing cryptographic operations on web content.

Measurements show that HTML content size does not increase significantly; rather, there is a reduction in size for text greater than 2KB. Cryptographic operations take only a few milliseconds to complete, and web content that contains both text and binary objects incurs a processing overhead that is less than 1 second for small sites and only a few seconds for larger sites.

We are motivated to work on this problem because we want to use webmail services without forfeiting the privacy of our messages, communicate with our financial institutions without having an intermediary learn our account information, and publish blogs with only a selected audience knowing what the content is. The protection offered by E2E cryptography at the application level is the correct model for these situations. W3Bcrypt is a step in the right direction for the privacy of end users.

References

1. V. Anupam and A. Mayer. Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies. In *Proceedings of the* 7^{th} *USENIX Security Symposium*, January 1998.
2. D. Brumley and D. Boneh. Remote Timing Attacks Are Practical. In *Proceedings of the* 12^{th} *USENIX Security Symposium*, August 2003.
3. N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell. Client–side Defense Against Web–based Identity Theft. In 11^{th} *Annual Network and Distributed System Security Symposium (NDSS 2004)*, February 2004.
4. C. Coarfa, P. Druschel, and D. S. Wallach. Performance Analysis of TLS Web Servers. In 9^{th} *Annual Network and Distributed System Security Symposium (NDSS 2002)*, February 2002.
5. D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proceedings of the* 10^{th} *USENIX Security Symposium*, August 2001.
6. R. W. et al. Platform for Privacy Preferences (P3P) Project. `http://www.w3.org/P3P/`, September 2005.
7. J. Gregorio. Secure RSS Syndication. `http://www.xml.com/pub/a/2005/07/13/secure-rss.html`, July 2005.
8. V. Gupta, D. Stebila, S. Fung, S. C. Shantz, N. Gura, and H. Eberle. Speeding Up Secure Web Transactions Using Elliptic Curve Cryptography. In 11^{th} *Annual Network and Distributed System Security Symposium (NDSS 2004)*, August 2004.
9. T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing. `http://www.w3.org/TR/xmlenc-core/`, 2002.
10. T. Imamura and H. Maruyama. Specification of Element-wise XML Encryption. `http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/att-0005/xml%enc-spec.html`, 2000.
11. S. Ioannidis and S. M. Bellovin. Building a Secure Web Browser. In *Freenix Annual Technical Conference (USENIX 2001)*, June 2001.
12. C. Lesniewski-Laas and M. F. Kaashoek. SSL Splitting: Securely Serving Data From Untrusted Caches. In *Proceedings of the* 12^{th} *USENIX Security Symposium*, August 2003.
13. H. Maruyama and T. Imamura. Element-Wise XML Encryption. `http://lists.w3.org/Archives/Public/xml-encryption/2000Apr/att-0005/01-%xmlenc`, 2000.
14. E. Rescorla, A. Cain, and B. Korver. SSLACC: A Clustered SSL Accelerator. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, August 2002.
15. B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the* 14^{th} *USENIX Security Symposium.*, pages 17–32, August 2005.
16. R. Sekar, C. Ramakrishnan, I. Ramakrishnan, and S. Smolka. Model-Carrying Code (MCC): A New Paradigm for Mobile-Code Security. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, September 2001.
17. Z. Ye and S. Smith. Trusted Paths for Browsers. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, August 2002.