Content-Based Security for the Web

Alexander Afanasyev
University of California, Los Angeles
aa@cs.ucla.edu

J. Alex Halderman
University of Michigan, Ann Arbor
jhalderm@eecs.umich.edu

Scott Ruoti
Brigham Young University
ruoti@isrl.byu.edu

Kent Seamons
Brigham Young University
seamons@cs.byu.edu

Yingdi Yu
University of California, Los Angeles
yingdi@cs.ucla.edu

Daniel Zappala
Brigham Young University
zappala@cs.byu.edu

Lixia Zhang
University of California, Los Angeles
lixia@cs.ucla.edu

ABSTRACT

The World Wide Web has become the most common platform for building applications and delivering content. Yet despite years of research, the web continues to face severe security challenges related to data integrity and confidentiality. Rather than continuing the exploit-and-patch cycle, we propose addressing these challenges at an architectural level, by supplementing the web's existing connection-based and server-based security models with a new approach: content-based security. With this approach, content is directly signed and encrypted at rest, enabling it to be delivered via any path and then validated by the browser. We explore how this new architectural approach can be applied to the web and analyze its security benefits. We then discuss a broad research agenda to realize this vision and the challenges that must be overcome.

CCS Concepts

•**Security and privacy** → **Web protocol security;** *Key management; Browser security; Usability in security and privacy;* •**Networks** → *Naming and addressing;*

Keywords

content-based security; web security; end-to-end encryption

1. INTRODUCTION

The World Wide Web is the most popular platform for building modern client-server applications and for delivering online content; it is a major factor in the Internet contributing over $1.6 trillion to the world economy every year [46]. Unfortunately, despite decades of research, the web platform continues to be plagued by security problems. Attacks on web servers result in the theft of passwords (e.g. [40, 44]) and confidential data (e.g. [5, 29, 68, 74]), resulting in billions of dollars in economic losses [58]. Vulnerabilities in TLS—the cryptographic protocol that underlies HTTPS—and its implementations have regularly and infamously undermined the security of millions of websites [1, 6, 12, 23, 38]. Classic web vulnerabilities, such as cross-site scripting (XSS), remain among the most commonly reported security problems worldwide [15, 57, 72]. Perhaps most alarmingly, nation-state attackers have recently begun injecting their own code into third-party websites to hijack visitors' bandwidth for distributed denial-of-service attacks [47]. Left unchecked, these security problems threaten to undermine the web's potential as an engine of commerce, communication, and economic growth.

Many of the security problems affecting the web are due to a fundamental mismatch between its semantics and those of existing defenses. The web is a *content-oriented* system by nature: HTTP has two types of messages, requests for content at a particular URL and responses containing the requested data. In contrast, the most important web security mechanisms apply either a *connection-oriented* or *server-oriented* architecture that focuses on protecting connections between the browser and particular servers. For example, HTTPS [61] provides a confidential and authenticated channel between browsers and servers, and the same-origin policy [11, 53] isolates content loaded from different servers on the client side. This approach was not the result of research but historical convenience, since the web is constructed on top of the TCP/IP protocol stack, which can only provide point-to-point connections between clients and servers; thus security was patched onto these connections.

This mismatch contributes to a spectrum of security problems. The threat model behind these defenses assumes that web servers are secure; consequently, attackers who compromise a server can arbitrarily intercept or modify the content contained there, and so servers become a highly desirable target. This problem is compounded because, due to the web's scalability and performance demands, web requests are increasingly being fulfilled by network intermediaries,

NSPW '16 September 26–29, 2016, Colorado, USA
© 2016 ACM. ISBN 978-1-4503-2138-9…$15.00
DOI: 10.1145/1235

such as content delivery networks (CDNs), caches, and other middleboxes [2,13,18,56][1]. Under today's secure channel model, these middleboxes are able to see and modify the data passing through them, so they too can be attacked to compromise data and websites. Moreover, HTTPS servers and middleboxes have to safeguard their private keys, yet these keys need to be kept accessible to the servers and middleboxes continuously in order to satisfy TLS connection requests.

At the same time, more websites are integrating resources from third parties, ranging from analytics tools to advertisements to user-generated content. Intermingling content from different sources must be accomplished under strict server-oriented security rules to prevent XSS and other attacks, but existing protections (such as the same-origin policy) provide only coarse isolation and are complicated for developers to use correctly [34]. Although the web's power comes from the rich interconnection of hyperlinks, where any content may reference any other, the server-oriented security model is simply incapable of expressing sophisticated semantic and security relations among content.

We propose to address the security challenges facing today's web applications using a fundamentally different approach: *content-based security.* This approach is influenced by our work with Named Data Networking [36,55,82], a clean-slate design of a new Internet architecture. Under a content-centric security model, each piece of web content can be signed for authenticity and encrypted for confidentiality independently of how it is delivered to end users. Policies about who can modify or access content, and about what executable content is allowed to do in the context of a site, can be tightly bound to the content itself, rather than merely to the server or domain at which it is hosted. Clients can verify the authenticity of items of content with fine granularity, potentially down to the level of individual user comments on a forum. Content-based security is an approach that is well suited to providing secure communication over a network that is increasingly considered hostile [4].

In this paper we explore the design space that is created by the principle that all content on the web should be secured, independent of (in addition to) the channel over which it is carried. This represents a new paradigm for web security. The principles of information-centric design have been well explored for network architecture, resulting in a wealth of research related to forwarding, routing, transport, and caching, with new frontiers still being explored for networks with intermittent connectivity and for the Internet-of-things. These principles have not yet been explored in the area of web security; this paper is a first exploration of the advances that can be made under this new paradigm.

We begin by analyzing the web's current security problems, focusing on limitations inherent in the web's architecture. We then argue that a web architecture with content-based security at its core can solve many of these problems. Finally, we lay out a research agenda that spans issues from trust management for content producers to secure cryptography in the browser, along with key management at both ends. Our work in this space is in an early, exploratory phase, so our designs at this point are preliminary in nature and reveal more questions than answers.

[1]Cisco predicts that browsers will get the majority of their content through CDNs by 2019 [16].

2. SECURITY PROBLEMS AFFECTING THE WEB

There is a fundamental mismatch between the web's content-centric architecture and the connection-oriented and server-oriented approaches taken by existing web security mechanisms. This leads directly to a number of limitations and weaknesses that contribute to many urgent security problems.

One family of examples stem from HTTPS, the cryptographic transport used to secure websites. HTTPS is an entirely connection-oriented protocol; the abstraction that it provides is a secure channel between the browser and server, with no awareness of the content itself. This results in a number of problems, including:

- Although HTTPS secures the connection from the server to the browser, servers themselves remain a central point for attack. Data is often stored unencrypted at rest, leaving it vulnerable once server security is broken.

- HTTPS servers authenticate themselves to browsers by using their private keys during the connection handshake. This requires the private key to be accessible to the front-end web server at all times, exposing it to theft or leakage via side channels.

- HTTPS makes it difficult to follow the principle of least privilege, since a single key is normally used to safeguard all connections to a domain name.

- Although computational advances have reduced the cost of deploying HTTPS encryption for traditional servers, it can still be prohibitive for low-power and low-resource embedded systems and for the growing "Internet of Things."

These issues are further compounded because, at the same time that the security community is pushing for wider adoption of HTTPS, a growing fraction of web content is being delivered via CDNs and middleboxes [16]. These intermediaries violate HTTPS's model of an end-to-end secure connection, exposing sites and their users to several problems:

- CDNs need to terminate HTTPS connections in order to provide their services, allowing them to see or modify data on its way from the server to the user. Compromise of a single CDN box can expose confidential user information, as content is completely unprotected outside the HTTPS channel.

- CDNs tend to hold the private keys for large numbers of servers, making them an extremely high value target for attackers. This dangerous concentration of key material is a symptom of the mismatch between HTTPS's connection-oriented security model and the modern web's content-oriented performance demands.

- Secure connections to CDNs or middleboxes can mask insecure back-haul connections. For instance, CloudFlare recently introduced free, one-click SSL support for all its customers [17], but this only protects the connection from CloudFlare to the browser. The connection from CloudFlare to the back-end web server often operates over unencrypted HTTP, exposing data to theft or tampering with no indication to the user. Middleboxes too can weaken HTTPS by failing to properly implement TLS or adequately validate certificate chains [43].

Another family of security problems stem from the web's server-oriented isolation model. Web applications enforce the *same-origin policy*, under which scripts running on one page can only access data contained on a second page if both pages were loaded from the same hostname and port. The same-origin policy maps poorly onto the content-oriented semantics of the web, making it both too inflexible and too coarse:

- The same-origin policy grants identical privileges to any script on a server, with no provision for assigning fine-grained levels of trust. This makes it extremely difficult to safely isolate user-generated content from other content on the same server, such as in the case of personal homepages on a university site or customizable profiles on a social network.
- Websites frequently incorporate scripts from external servers, for purposes such as advertising and analytics, and these scripts gain unrestricted access to any data within the site's origin. If the external server is compromised, it can deliver a malicious version of the script that attacks all sites that use it.
- Within a page, the same-origin policy has no concept of who originated each piece of data. This leads to frequent cross-site scripting attacks (XSS): an attacker embeds a malicious script by posting data that gets incorporated into the content of the page, allowing it to be executed inside the site's origin.

The urgency of these problems is reflected in several recent World Wide Web Consortium proposals that attempt to mitigate them. For instance, Content Security Policy [79] defends against XSS using an HTTP header that specifies a whitelist of approved sources from which a page may load content—a simple form of reduced privilege. Subresource Integrity [3] defends against attacks that tamper with externally hosted scripts by letting a page specify their hashes—a simple form of control-centric integrity. These defenses point in a promising architectural direction, but they do not go far enough. In this paper, we introduce a comprehensive approach to fine-grained and content-centric security that has the potential to mitigate all of the problems described above without introducing unnecessary complexity to users, developers, or system administrators.

Although previous systems research has touched on ways to secure web content directly, there has yet to be a systematic architectural exploration of this approach. SSL Splitting [41], SINE [27], HTTPI [14], HTTPi [69], and iHTTP [28] introduce a variety of cryptographic mechanisms for efficiently providing content integrity in the face of hostile networks and malicious intermediaries, but all focus on the familiar model of proving that content came from a particular server. Spork [52] goes further, using remote attestation to allow the client to verify that the server was operating properly when it generated the content. Closest to our approach is WebTrust [8], a recently proposed framework that combines finer-grained cryptographic integrity protections with a clean separation between servers and content generators. While WebTrust shows one way that a content-based approach to integrity can be made workable from a cryptographic perspective, there is far more to be done to understand how to build a comprehensive content-based security architecture for the web that supports both integrity and privacy while being usable for publishers and end-users alike.

3. CONTENT-BASED SECURITY

In this section we argue that the drawbacks of the web's connection-based and server-based security models can be addressed by using principles developed for information-centric network architectures. We first provide an overview of the Named Data Networking (NDN) [36, 55, 82] architecture, which provides content-based security at the network layer. We then analyze how this approach can be used to address many of the web's security problems.

3.1 Overview of NDN

NDN is a new network architecture that supports content-based networking as a fundamental part of the network layer[2]. A host requests content by sending an *interest packet*, which specifies the name of the desired content, and the network responds by sending back a *data packet* containing the requested content. Since the requester only specifies what it wants (i.e., the data name), the network has the freedom to make intelligent decisions on where to forward an interest packet. It could be satisfied using an available replica of the data hosted by the original data producer or by a third-party storage provider, or even by an in-router cache containing the requested data.

Figure 1 illustrates the basic operation of an NDN network using the New York Times website as an example. NDN names are structured hierarchically and can refer to any piece of content—a chunk of a video that is being streamed, a portion of an article, or even an actuation command. For example, data for an NY Times article about NDN may have the name "`/nytimes/tech/2015/08/20/ndn`", where '/' delineates name components in text representations, similar to URLs. When the article about NDN is ready to be published, the website editor creates data packet(s) with the content of the article, splitting content into multiple segments if necessary. Each piece of NDN data has a unique name and is signed at the time of its production, cryptographically binding the name and the data. Therefore, NDN data packets can be stored and retrieved from anywhere in the network. For content that may change dynamically, the name must include additional components to disambiguate different versions. In our example, the NY Times could add a versioning component, such as a hash of the content, a timestamp indicating when the last revision was made, or simply a version number (e.g., "`/nytimes/tech/2015/08/20/ndn/_v=42`").

The website makes its content reachable in the network by announcing its name, "`/nytimes`", to the global routing table. A consumer that wants to view the NDN article sends an interest packet to the network (e.g., "`/nytimes/tech/2015/08/20/ndn`"), with metadata indicating that the latest version is requested. The interest is forwarded through

[2]The original principles of Content-Centric Networking (CCN) were first described by Van Jacobson in 2006 [35]. When the National Science Foundation funded the work in 2010, the project was renamed Named Data Networking (NDN) [82].

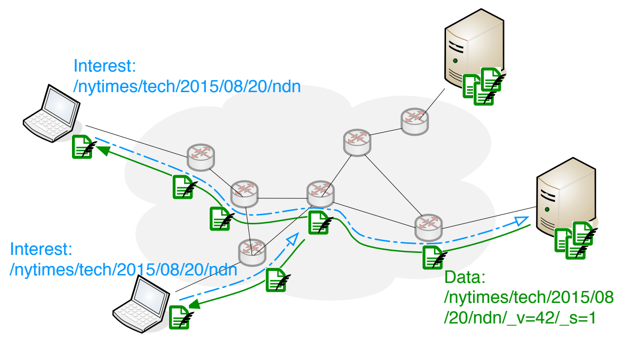


Figure 1: Example of NDN communication

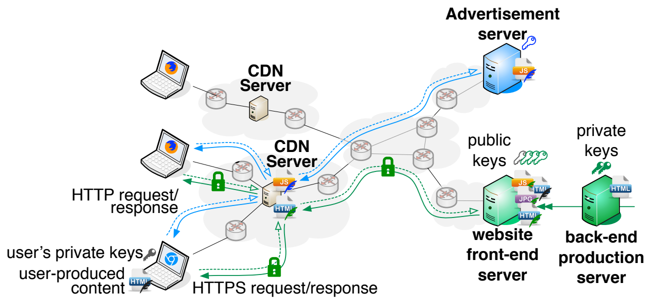


Figure 2: Content-based security for the web

the network to the nearest replica of the website. Once the interest packet meets a data packet with the matching name, the network returns the data packet back to the original requester or requesters along the reverse path of the interest packet. The data packet carrying the article can be cached by routers along the path, so that when a router receives another interest for the same article, it can immediately satisfy the interest with the cached copy.

In NDN every named piece of content (data packet) must be signed. This ensures that the data can be authenticated regardless of how/where it is retrieved. Besides the signature, each data packet also carries additional metadata including the signing key name. To authenticate a data packet, one needs a trust model that defines which keys are authorized to sign which data (trust rules) and one or more trusted keys to bootstrap the trust (trust anchors). Any entity—applications, dedicated network storage elements, and even network routers—that learns the trust model for a given piece of content can verify its authenticity, and may perform necessary actions when the authentication fails (e.g., discard the packet, or try an alternative path to retrieve). Keys in NDN are just another type of data, thus they also have unique names and can be fetched and authenticated in the same way as any other data packets; data packets carrying public keys are effectively NDN certificates.

NDN can also support content confidentiality by encrypting content at the time of production. The encrypted data packets can be stored anywhere in the network as needed, and delivered to requesters through any path without compromising their confidentiality. Only the authorized parties are given the correct decryption keys to access the original content.

3.2 A Content-Based Architecture for the Web

We believe that NDN's content-oriented security model is a natural fit for securing the web. Both NDN and the web use a hierarchical namespace that is application-dependent, deliver data associated with these names using a request–response model, and cache content to improve performance.

Figure 2 explores a design for integrating content-based security into the web. A producer generates content on a back-end web server, signing and potentially encrypting the content with its private keys. It then uploads this content onto its front-end server. A web browser issues an HTTP request for the content and is directed to a CDN server. This service fetches the signed (and possibly encrypted) content, loads the content into is cache, and delivers the content to the browser. Any additional content, such as third-party JavaScript or advertising, is likewise loaded from additional servers. This content is signed using authorized keys from those domains. Finally, any user generated content, such as a payment details or social networking posts, is signed and possibly encrypted by the user's authorized keys. Note that each of the paths shown in this figure can be additionally secured using an HTTPS connection. When content is only signed, and not encrypted, this provides confidentiality during transport.

Bringing the advantages of content-based security to the web requires supporting content signatures and encryption within HTTP. Although there are ongoing research efforts to add content-based security semantics into HTTP, the provided solutions are largely piece-meal, such as the standardization effort [10] to define data structures and encoding formats for signed and encrypted JSON objects. We envision adding relevant HTTP headers, such as a header that contains the signature and another that provides the validation key name. Note that keys are simply another piece of named content.

An important feature of NDN is that content is immutable, so a unique name must be used to refer to content that does not change once it is signed. In HTTP, on the other hand, URIs are usually not unique, rendering different content if requested at different times or using different HTTP request headers (cookies, accepted language, etc.). We propose additional syntax and semantics be added to URIs to define persistent names, such as using parameters to provide version numbers or other context that converts a URL into a unique name.

While signing every piece of content appears to impose a high overhead, we believe this approach is feasible. Many web pages are static, and even dynamic pages are often composed from largely static elements. For example, the Amazon home page consists of numerous elements, but nearly all of them are static HTML, images, CSS, or Javascript. Likewise, validating signatures for many content items is feasible, even on mobile devices. Encryption operations on mobile devices are currently accelerated, with typical devices able to stream HD movies from Yahoo over encrypted connections. The approach is even feasible with today's modern web applications, which are often highly dynamic, such as video chatting and mapping. Dynamic services ultimately deliver data—voice or video packets, map data—and this could all be signed and potentially encrypted on the fly, as it is with current TLS connections.

We have built several prototypes to experiment with this architecture, including a web server that inserts content signatures in HTTP headers and a Chrome extension that validates the signatures. Because Chrome extensions cannot access the body of a response, the validation is done via a SOCKS proxy. Our next prototype will integrate validation directly into the Chrome browser.

3.3 Analysis of Benefits

Based on our experiences with the NDN architecture and with content-based security prototypes for the web, we believe bringing content-based security to the web would provide the following benefits:

- *All content is signed, providing both integrity and authentication directly for the content.* Web pages can be signed by the content-creator's key, then delivered by CDNs over an encrypted channel that is protected by the CDN's separate key. This avoids the current dangerous situation where CDNs must use each site's private key for its content, concentrating high-value key material in one location. This also solves the problem of CDNs transferring content among their back-end servers using unencrypted connections. Finally, this provides a way to combat cross-site scripting attacks, since the browser would only run scripts that are signed by the originating domain or an authorized third party.

- *Sensitive content is encrypted at rest.* It is not possible to steal or tamper with content simply by breaking into the server where it is housed. This is particularly important because content is increasingly being hosted at CDNs. While this can be accomplished today, it must be done on a per-application basis. Because so many applications are built on the web, resulting in a new narrow waist for the Internet [59], we gain a great deal of leverage by building content-based encryption into the web.

- *Private keys are kept offline.* Static content is signed and encrypted offline before being placed on a server or CDN. This allows private keys to be kept offline, where they have much stronger protection.

- *The principle of least privilege is followed.* Currently web content is generally protected by a single key for each server. With content-based security, a hierarchy of keys is established, enabling different sets of resources within a site to be protected by independent keys. This can significantly limit the damage from key compromise.

- *Embedded systems can host secure content.* Content that rarely changes could be protected more efficiently by signing and encrypting it once, rather than every time it is requested.

4. RESEARCH AGENDA

Our research agenda to validate these ideas spans a variety of issues from content creation on the back end to content validation in browsers, including trust management, support for cryptography in the browser, key management for both content providers and users, and overall usability studies.

4.1 Trust Management

An important question for content-based security is which keys to trust when validating signed and encrypted content. This is a critical weakness in the current Internet; the Certificate Authority (CA) system requires browsers to trust a very large number of authorities by default (1,832 signing certificates, in one recent study [24]), and any of these certificates can sign for any domain. Thus the authentication guarantees provided by the system are only as strong as the weakest CA. This weakness has been exploited multiple times; for example, in 2011 DigiNotar's servers were hacked and more than 500 certificates were fabricated by the intruder, including a certificate for Gmail that allowed the intruder to access stored email for 300,000 Iranians [37]. This happened despite the fact that Gmail does not use DigiNotar to sign its certificates. This problem is exacerbated by CAs that do not follow best practices [22, 48] and governmental ownership and access to CAs [24, 71].

Content-based security provides a way to upgrade this security so that a content provider can indicate which keys may sign for different portions of the namespace it controls. In addition, a provider can securely link to content, so that any navigation from the provider's namespace to another is also done securely. As content-based security spreads, it thus creates a separate web of stronger trust, overlayed on the existing web.

This relationship between the URI of the content and the URI of its associated keys—which keys are valid for which parts of the namespace—can be formalized in a trust schema [81]. A trust schema includes a set of linked trust rules and one or more trust anchors. For example, the namespace for a tech blog (`/nytimes/tech/blog/*`) could be linked to the anchor for the editor (`/nytimes/editor/tam/KEY/1`), who can sign keys for individual authors. A generalized syntax allows further refinements.

This notion of a trust schema frees the web from its restrictive channel-based security model, which restricts trust to content from the same origin [53], or from different origins with a coarse (e.g., end-host, content type) granularity [79]. For example, a web page may need to include some specific advertisement scripts from an advertisement website, while excluding all the other scripts from the same website. Existing attempts to address this problem introduce a variety of cryptographic mechanisms for efficiently providing content integrity in the face of hostile networks and malicious intermediaries [14, 27, 28, 69], but all focus on the familiar model of proving that content came from a particular server. With a trust schema, on the other hand, a site that includes advertising or other content from third parties can provide a separate trust anchor and hierarchy of permitted keys for each third party. User-provided content such as comments on a blog can be relegated to being self-signed or signed by a separate trust anchor, so that browsers know to treat it differently.

For the trust schema to be validated by the browser, it must depend on one or more trust anchors, similar to a web site needing its key to be signed by some key in the browser's root store. An open question is how to ensure that trust anchors can be distributed reliably to browsers without the weaknesses of the CA system. One possibility is to adopt some of the mechanisms being proposed to strengthen the CA system, such as Certificate Transparency [66]. A variety of alternative distribution methods are possible that would allow greater freedom from the CA system, including models that follow the strict naming hierarchy, such as DANE [9, 32], a public certificate log [66], pinning [25, 49], and evidentiary trust models such as Perspectives [78]. To explore these alternatives, we are building a certificate authentication platform that makes it easy to develop and deploy new authentication systems, as well as aggregate results among them. Of particular concern is the scalability of these authentication systems to large numbers of sites and anchors, especially if individual users (or their devices) sign the content they generate. It may be possible to gain increased security by distributing trust among many anchors, provided the compromise of a single anchor doesn't grant significant privileges.

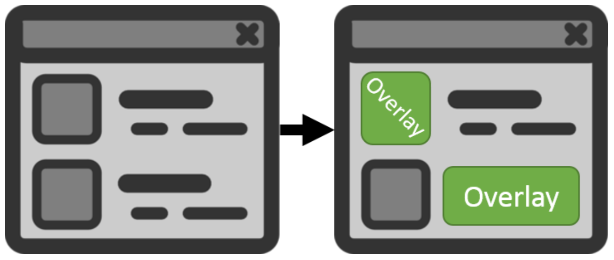
Another issue relates to security indicators used in the browser to represent whether content has been signed correctly. This is a longstanding issue for secure content, with research studying how best to communicate the security of a TLS connection in the browser through the classic lock icon [26]. One open question is whether a failed signature should be communicated differently from an invalid or expired certificate, and what action users would be expected to take. Another is whether the user should be alerted when some portions of a page have valid signatures but others do not. This is analogous to how browsers currently alert users when some content on a page is insecure—it is not clear whether these warnings are actionable for users. These are some of the numerous usability issues that need to be studied (see Section 4.5).

Another issue relates to the timeliness of content: a browser needs to get the most recent content for a page, and avoid the possibility of a man-in-the-middle providing signed but stale content. This is currently handled on the web by relying on TLS to ensure the browser is connected to the appropriate server. As mentioned in Section 3.1, NDN content names should include versioning information, and a content request can specify that it desires the latest version. With content-based security, the browser could use similar versioning in the names (URIs) or in HTTP headers, and the browser could use HTTP headers to request a response that includes a recent timestamp (e.g. within the past minute).

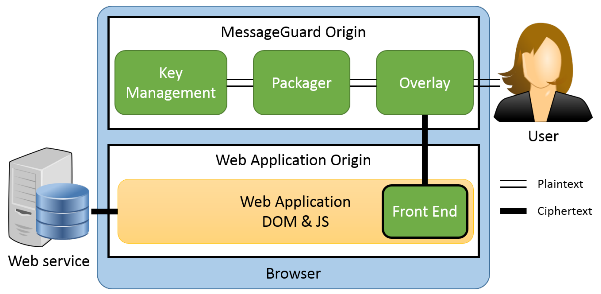
Finally, an interesting area to pursue is interaction between the trust schema and existing privilege separation mechanisms that regulate interactions among various JavaScript applications. There is a wide variety of work in this area, including object capability systems [21,45,51] and those that rely on aspect oriented programming [50,75], or various other techniques [19,20,33,73]. When the trust schema is validated and enforced by the browser, these decisions should inform the privilege separation mechanism so that untrusted code is properly sandboxed if it is allowed to run.

4.2 Supporting Cryptography in the Browser

Users will need end-to-end encryption to sign and encrypt their own content in a variety of scenarios. At a minimum, content uploaded from users must be signed with their public key, or a delegated key if the user manages a hierarchy of keys. Sensitive information, such as credit card information for a financial transaction, must also be encrypted with the appropriate server's key. Likewise, users need end-to-end encryption to protect their data from honest-but-curious service providers, from governments that coerce or cooperate with service providers, and from miscreants who break into service providers. For example, secure messaging and secure webmail apps use end-to-end encryption between users, while the site that provides the messaging or webmail service is not necessarily trusted, even though it may have a strong



(a) Concept: Portions of the web application (left) have been overlayed with secure interfaces (right).



(b) Architecture: Plaintext is present only within the MessageGuard origin, preventing the web application from ever accessing it.

Figure 3: MessageGuard

reputation. This has been an area of active development recently, with webmail services such as ProtonMail [60] and Virtu [77] providing encrypted email in the browser, and other software focusing on instant messaging.

Meeting this challenge requires support for cryptography in the browser that guarantees user privacy from code that may not be completely trusted with personal information. The browser needs a method for signing/validating and encrypting/decrypting data so that the web service sees only ciphertext. We particularly need systems that can provide ubiquitous support for any site on the web, rather than those that require special configuration for each site.

A variety of approaches have been used in this area, including copy-and-paste from an external app into the browser [65], using a ShadowDom [30], and using IFrames [73]. We have found several fundamental security flaws with the ShadowDom approach [64], so we have focused on on using IFrames to provide a secure space that cannot be accessed by code downloaded from the origin web server. We have used this approach to develop a system called MessageGuard that provides ubiquitous encryption for all web applications [64]. As shown in Figure 3a, MessageGuard uses secure overlays to provide a generic interface for encrypting web content that is uploaded in form fields, with customized interfaces available on a per-application basis. We have used MessageGuard to implement Private WebMail, which uses security overlays to tightly integrate with webmail services like Gmail [63].

Figure 3b shows the architecture of MessageGuard. To provide secure overlays, MessageGuard uses IFrames, which allow an HTML document from the overlay's origin to be displayed as part of an HTML document from the web application's origin. Due to the same-origin policy used by browsers, content in the overlay will be inaccessible to the web application [53]. Communication between the overlay and the web application occurs through the web messaging API [31]. As long as the overlay never passes plaintext data to the web application and never executes JavaScript sent to it from the web application, it will remain secure. Other approaches have used the ShadowDom [30], but this approach has several security flaws, illustrating the danger of building on a mechanism this is not intended to provide security features. Using IFrames avoids this problem, because it is explicitly intended to be a security mechanism for the browser; venders pay significant attention to patching vulnerabilities quickly when they are found.

Another approach that is complementary to MessageGuard is to implement secure cryptographic operations and key storage as an external component, separate from the browser. This can strongly isolate keys from the browser, so that any vulnerabilities in the browser do not compromise the user's keys. Web Cryptography (WebCrypto) API [70] is an ongoing standardization work at W3C to define a JavaScript API for securely performing common cryptographic operations such as hashing, signing, verification, encryption and decryption. However, there are several critical limitations of the WebCrypto API: web applications can still gain full access to the decrypted messages and existing implementations of secure storage are browser-specific [54]. An open area for research is in cross-browser/cross-device security environments where private keys may be stored in a variety of ways—within a local database, via a flash drive, through devices connected via bluetooth, etc.

4.3 Key Management for Content Providers

Transitioning the web to content-based security will put a premium on key management, a central challenge for applied cryptography and usable security. Content providers need mechanisms to manage the key lifecycle, including generating keys, establishing a hierarchy of signing authority, establishing expiration periods, and revoking keys that have been compromised. They likewise need assistance developing trust schemas that express desired security properties.

Automation will play a key role in helping providers manage security policies that follow the least-privilege principle. It is highly desirable to limit the scope of keys in order to limit exposure of cryptographic keys and reduce the damage of key compromise. Our experience developing the automated *Let's Encrypt* certificate authority [7, 42] provides a path to explore automatic key generation and maintenance in the context of a hierarchical trust schema. With Let's Encrypt, a system administrator runs software that automatically validates ownership of a domain by meeting challenges, such as adding a DNS record or publishing an HTTP resource under a well-known URI. Once the challenges have been satisfied, the certificate authority issues a certificate, which is automatically configured at the site's web server. Certificate renewal and revocation are accomplished via simple commands, again with the process automated.

We envision adapting this process to provide automated management of complex hierarchies of keys with their associated privileges. A system administrator identifies the key principals (authors, editors, web designers, user experience engineers, etc.) and their roles, and notifies the system when these roles change over time. With this input, the system can automatically update a trust schema and generate/update keys for the roles as needed. Likewise, the administrator may need to identify keys that have been compromised, using input from intrusion detection systems.

A variation from the certificate revocation model is to issue short-lived certificates. This obviates the need for revocation and its associated challenges. The scope of a compromise is limited because of the short expiration period. With suitable automation, the overhead for frequent (e.g, daily, weekly) certificate issuance may be acceptable.

4.4 Key Management for Users

Key management is an especially important challenge for users in a web with content-based security. To decrypt content being received from a server, users need help identifying which keys are valid. Much of this can be automated through the use of a trust schema, similar to how TLS is automated today. However, when signing content with their own key, for transmission to a server, users need help managing their identity among the various devices they own. Likewise, users need assistance when authenticating keys for other users.

Unfortunately, we have been stuck in a decades-long situation where "Johnny can't encrypt" [80]. For secure email, our ongoing research indicates that users have significant success encrypting email using an automated mechanism based on Identity-Based Encryption (IBE), which removes many of the responsibilities from users but requires trust in a third party to store and manage keys [62, 63]. IBE enables users to immediately interact with each other, for example to send email to a user who has not yet established a public key with their identity. IBE also enables users to easily recover their keyed data (and thus all their past encrypted data) if they lose a password or other identifying material. Manual mechanisms, such as PGP, require users to manage their own keys, potentially with help from well-designed software. This approach provides increased security and limited risk by removing any trusted third parties, but leads to other challenges such as securely transferring keys between devices and the lack of key recovery if the user loses the key or forgets the password protecting the key. Moreover, there has been a significant lack of well-designed software that helps ordinary users effectively manage their own keys [67, 80].

One avenue for more usable key management is key escalation, where novice users are first introduced to secure content with a key escrow system that creates and manages their keys for them. Once users are engaged with the system, client-side software can generate a public/private key pair for the user and help them transition to browser-based tools and mobile applications for key management as they gain more experience.

Using key escalation will enable users to avoid relying on a trusted third party to store their keys, but must address the usability challenges that plague current key management software. Key creation on a single device is relatively simple, and with support from a system library users can easily use this same key on different browsers. To transfer this key to other devices, there are a number of approaches, each with deployability, security, and usability tradeoffs. One design could store the key on a trusted server, encrypted with a password, similar to LastPass [40]. This will be helpful for users who are familiar with password-style authentication. Other designs could transfer keys among devices via bluetooth or a flash drive. To protect users from loss, a master key could be kept offline on a flash drive, with delegated keys assigned to devices. This would allow for expiration or revocation of keys if a device or key is stolen.

Finally, users also need methods for authenticating keys for each other. This covers cases where users directly communicate with each other or via a third party service, such as email or instant messaging. PGP has classically relied on a web of trust, but most users do not have the expertise needed to participate in a key signing party.

Keybase offers an interesting alternative by mapping public keys to social media identities [39]. Their system asks you to respond to a challenge by posting on your account at Twitter, GitHub, Reddit, CoinBase, or Hacker News, or on your personal web site. Keybase includes a number of methods for key management, including key escrow and browser scripts (simple to use but less secure), a program they provide for automating key generation and proofs (somewhat more difficult to use, but more secure), to a recipe of shell instructions in case you do not trust their software (most difficult to use, but most secure). Once you have uploaded your public key, other users can retrieve your key using your account name as an identifier. Keybase does not provide an easy migration path from key escrow to more secure key management methods, and the tool used for managing your own keys requires familiarity with the command line. Hence, there is substantial work to be done to further improve on social methods to validate keys.

We are studying an approach where the issues of key discovery and validation can be solved by bootstrapping off the trust that already exists among users of online social networks (OSNs) [76]. Each OSN already provides users with long histories of posts, pictures, and personal communications from their contacts and provides authentication of its users (via password logon). By following a verified user on Twitter or accepting a friend request on Facebook, users are already making an authentication judgment. Thus if public keys were posted to and associated with users' various OSN accounts, an organic set of verified key-identity pairs could emerge. By querying keys for a user from multiple OSNs and checking for agreement, the application could enhance trust in a public key, as multiple OSNs would have be compromised or collude to present a believable false key. Such a system mitigates much of the manual key validation problem as users can rely on the robustness of multiple authorities vouching for the authenticity of a particular key. This mechanism also has an added benefit—users need only have some type of OSN account (or phone number) to be discovered, rather than forcing every user to have an account with a specific service or OSN.

4.5 Usability

Usability must be addressed for content-based security to be successfully deployed. Our prior work with secure webmail indicates that careful design can improve usability significantly [63]. Four design choices helped in this case: (1) using an artificial delay[3] to show users their message being encrypted enhances user confidence in the strength of the message encryption while simultaneously instructing users on who can read encrypted messages; (2) using email composition interfaces to show users which emails are encrypted helps them avoid mistakenly sending sensitive information in the clear, reducing the mistake rate from 10% to 2% in our lab experiments; (3) including contextual clues to help users understand how to use secure email correctly; and (4) implementing inline, context-sensitive tutorials to instruct users, improving view rates for tutorials from less than 10% to over 90%.

[3]We believe this delay can be eliminated over time as users become adapted to the system. There are also numerous possible alternatives, such as showing ciphertext alongside plaintext, using explanatory text in the composition interface, etc.

A broad range of usability methods can be applied to evaluate solutions for a content-secure web. This process begins with surveys and interviews to better understand user preferences and attitudes toward security. Cognitive walkthroughs and heuristic evaluation are then used during the prototyping phase to emphasize usability from the perspective of the user. Interface design needs to ensure that users know what actions are available to them, that these actions will be appropriate for the effect they are trying to achieve, and that they receive positive feedback after taking an action. Lab usability studies provide feedback on more complete systems, gathering both quantitative and qualitative user feedback. Finally, long-term studies of use in the wild validate that user needs are in fact being met.

Usability studies are necessary to evaluate whether administrators can create effective trust schemas, content authors can correctly sign their data, users can take effective action based on warning messages when data validation fails, users can properly use a public/private key pair to manage their identity across devices, and so forth.

There are a number of usability challenges to address in content-based security. (1) What security features can be hidden from users and what features need to be visible in order to provide users with an accurate model of the security that is being provided? (2) When a problems occurs, what notification is effective at informing users what is happening and leading them to make appropriate decisions to address the problem? (3) When content is loaded from multiple web servers and some of it fails verification, how is this communicated to users?

5. CONCLUSION

Bringing content-based security to the web will make it easier for content providers to publish secure content that can be delivered over any path and automatically validated by browsers. This approach can solve many of the problems that plague today's web. Signing content mitigates man-in-the-middle and cross-site scripting attacks, and encrypting data at rest enables content to be protected from theft regardless of where it is hosted. Private keys can be kept offline and the principle of least privilege can be followed.

A number of research issues must be solved to make this new approach a reality, and we have identified the following major problems. Content providers need automated trust management for a hierarchical collection of authorized keys and the relationships among their data and third-party scripts. Browsers need methods for securely integrating cryptography so that the trust schema published by a site is followed and so that a user's keys and private data are isolated from untrusted JavaScript. Users need help managing their identity and authenticating the identities of other users. We are in the process of characterizing the performance impacts of the architecture. There may well be other issues that we have missed; we are looking forward to the community's input to help move this research agenda forward.

Acknowledgements

We're grateful for feedback from the reviewers and our shepherd, Sarah Meiklejohn, and for formative discussions with kc claffy and other members of the NDN project.

This material is based in part upon work supported by the National Science Foundation under grants CNS-1345254, CNS-1409505, CNS-1345318, CNS-1518888, and CNS-1528022, and by an Alfred P. Sloan Foundation Research Fellowship. This material is also based in part on research sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD) via contract number HHSP233201600046C. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

6. REFERENCES

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security (CCS)*, Oct. 2015.

[2] Akamai. Akamai website. https://www.akamai.com/. Accessed: September 23, 2015.

[3] D. Akhawe, F. Braun, F. Marier, and J. Weinberger. Subresource integrity. http://www.w3.org/TR/2015/WD-SRI-20150916/, Sept. 2015. Accessed: September 23, 2015.

[4] J. Angwin, J. Larson, C. Savage, J. Risen, H. Moltke, and L. Poitras. NSA spying relies on AT&T's 'extreme willingness to help'. https://www.propublica.org/article/nsa-spying-relies-on-atts-extreme-willingness-to-help, 2015. Accessed: September 18, 2015.

[5] Anthem. Statement regarding cyber attack against Anthem. https://www.anthem.com/health-insurance/about-us/pressreleasedetails/WI/2015/1813/statement-regarding-cyber-attack-against-anthem, 2015. Accessed: September 23, 2015.

[6] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS with SSLv2. In *25th USENIX Security Symposium*, Aug. 2016.

[7] C. Babcock. 'Let's Encrypt' will try to secure the Internet. InformationWeek, 2015.

[8] M. Backes, R. Gerling, S. Gerling, S. Nürnberger, D. Schröder, and M. Simkin. WebTrust—a comprehensive authenticity and integrity framework for HTTP. In *12th International Conference on Applied Cryptography and Network Security (ACNS)*, volume 8479, pages 401–418, 2014.

[9] R. Barnes. DANE: Taking TLS authentication to the next level using DNSSEC. *IETF Journal*, 2011.

[10] R. Barnes. Use cases and requirements for JSON object signing and encryption (JOSE). RFC 7165, 2014.

[11] A. Barth. The web origin concept. RFC 6454, Dec. 2011.

[12] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *36th IEEE Symposium on Security and Privacy*, pages 535–552, 2015.

[13] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. RFC 3234, Feb. 2002.

[14] T. Choi and M. G. Gouda. HTTPI: An HTTP with integrity. In *20th International Conference on Computer Communications and Networks (ICCCN)*, 2011.

[15] S. Christey and R. A. Martin. Vulnerability type distributions in CVE. https://cwe.mitre.org/documents/vuln-trends/index.html, 2007. Accessed: September 23, 2015.

[16] Cisco. Cisco visual networking index: Forecast and methodology, 2014-2019. White Paper http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, 2015. Accessed: September 23, 2015.

[17] CloudFlare. CloudFlare one-click SSL. https://www.cloudflare.com/ssl. Accessed: September 23, 2015.

[18] CloudFlare. CloudFlare website. https://www.cloudflare.com/. Accessed: September 23, 2015.

[19] D. Crockford. Adsafe. http://www.adsafe.org/.

[20] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. Flowfox: a web browser with flexible and precise information flow control. In *19th ACM Conference on Computer and Communications Security (CCS)*, pages 748–759. ACM, 2012.

[21] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[22] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *13th ACM Internet Measurement Conference (IMC)*, 2013.

[23] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of Heartbleed. In *14th ACM Internet Measurement Conference (IMC)*, 2015.

[24] P. Eckersley and J. Burns. The (decentralized) SSL observatory. Invited talk at 20th USENIX Security Symposium, 2011.

[25] C. Evans and C. Palmer. Certificate pinning extension for HSTS. http://tools.ietf.org/html/draft-evans-palmer-hsts-pinning-00. Accessed: March 22, 2013.

[26] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo. Rethinking connection security indicators. In *12th Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association, June 2016.

[27] C. Gaspard, S. Goldberg, W. Itani, E. Bertino, and C. Nita-Rotaru. SINE: Cache-friendly integrity for the web. In *5th IEEE Workshop on Secure Network Protocols (NPSec)*, pages 7–12, 2009.

[28] J. Gionta, P. Ning, and X. Zhang. iHTTP: Efficient authentication of non-confidential HTTP traffic. In *10th International Conference on Applied Cryptography and Network Security*, pages 381–399, 2012.

[29] D. Grandon. Ashley Madison, a dating website, says hackers may have data on millions. http://www.nytimes.com/2015/07/21/technology/hacker-attack-reported-on-ashley-madison-a-dating-service.html, 2015. Accessed: September 23, 2015.

[30] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song. ShadowCrypt: Encrypted web applications for everyone. In *21st ACM Conference on Computer and Communications Security (CCS)*, pages 1028–1039, 2014.

[31] I. Hickson. HTML5 web messaging. http://www.w3.org/TR/2015/REC-webmessaging-20150519/. Accessed September 23, 2015.

[32] P. Hoffman and J. Schlyter. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC 6698, 2012.

[33] L. Ingram and M. Walfish. TreeHouse: JavaScript sandboxes to help web developers help themselves. In *2012 USENIX Annual Technical Conference*. USENIX Association, 2012.

[34] C. Jackson and A. Barth. Beware of finer-grained origins. In *Web 2.0 Security and Privacy (W2SP)*, 2008.

[35] V. Jacobson. A new way to look at networking. https://www.youtube.com/watch?v=oCZMoY3q2uM, 2006.

[36] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *5th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2009.

[37] G. Keizer. Hackers spied on 300,000 Iranians using fake Google certificate. Accessed: 27 October, 2015.

[38] G. Keizer. Apple's OS X 'Rootpipe' patch flops, fails to fix flaw. http://www.computerworld.com/article/2912619/mac-os-x/apples-os-x-rootpipe-patch-flops-fails-to-fix-flaw.html, 2015. Accessed: September 23, 2015.

[39] Keybase. https://keybase.io/. Accessed: September 23, 2015.

[40] LastPass. LastPass security notice. https://blog.lastpass.com/2015/06/lastpass-security-notice.html/, 2015. Accessed: September 23, 2015.

[41] C. Lesniewski-Laas and M. F. Kaashoek. SSL splitting: Securely serving data from untrusted caches. *Computer Networks*, 48(5):763–779, 2005.

[42] Let's Encrypt. https://letsencrypt.org/. Accessed: September 23, 2015.

[43] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu. When HTTPS meets CDN: A case of authentication in delegated service. In *35th IEEE Symposium on Security and Privacy*, pages 67–82, 2014.

[44] LinkedIn. An update on LinkedIn member passwords compromised. http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/, 2012. Accessed: September 23, 2015.

[45] S. Maffeis, J. C. Mitchell, and A. Taly. Object capabilities and isolation of untrusted web applications. In *31st IEEE Symposium on Security and Privacy*, pages 125–140. IEEE, 2010.

[46] J. Manyika and C. Roxburgh. The great transformer: The impact of the internet on economic growth and prosperity. McKinsey Global Institute report, 2011. http://www.mckinsey.com/industries/high-tech/our-insights/the-great-transformer.

[47] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson. An analysis of China's "Great Cannon". In *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2015.

[48] M. Marlinspike. SSL and the future of authenticity. *Black Hat USA*, 2011.

[49] M. Marlinspike and T. Perrin. Trust assertions for certificate keys. Internet Draft, 2012. https://tools.ietf.org/html/draft-perrin-tls-tack-00.

[50] L. Meyerovich and B. Livshits. ConScript: Specifying and enforcing fine-grained security policies for JavaScript in the browser. In *31st IEEE Symposium on Security and Privacy*, pages 481–496, 2010.

[51] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja: Safe active content in sanitized JavaScript. http://google-caja.googlecode.com/files/caja-spec-2008-01-15.pdf, Jan. 2008.

[52] T. Moyer, K. Butler, J. Schiffman, P. McDaniel, and T. Jaeger. Scalable web content attestation. *IEEE Transactions on Computers*, 61(5):686–699, 2012.

[53] Mozilla. Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. Accessed September 23, 2015.

[54] Mozilla. SubtleCrypto. https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto. Accessed: September 23, 2015.

[55] NDN Team. Named Data Networking (NDN) Project. Technical Report NDN-0001, Named Data Networking Project, Oct. 2010. http://named-data.net/wp-content/uploads/TR001ndn-proj.pdf.

[56] Netflix. Netflix Open Connect. https://openconnect.netflix.com/. Accessed: September 23, 2015.

[57] OWASP. OWASP top 10 project. https://www.owasp.org/index.php/Top_10_2013-Top_10, 2013. Accessed: September 23, 2015.

[58] Ponemon Institute. 2015 cost of data breach study: Global analysis, May 2015. http://www-03.ibm.com/security/data-breach/.

[59] L. Popa, A. Ghodsi, and I. Stoica. HTTP as the narrow waist of the future Internet. In *9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[60] ProtonMail. Protonmail. https://prontonmali.com/. Accessed: April 29, 2016.

[61] E. Rescorla. HTTP over TLS. RFC 2818, May 2000.

[62] S. Ruoti, J. Andersen, S. Heidbrink, M. O'Neill, E. Vaziripour, J. Wu, D. Zappala, and K. Seamons. "We're on the same page": A usability study of secure email using pairs of novice users. In *34th ACM Conference on Human Factors and Computing Systems (CHI)*, San Jose, CA, 2016. ACM.

[63] S. Ruoti, J. Andersen, T. Hendershot, D. Zappala, and

K. Seamons. Private Webmail 2.0: Simple and easy-to-use secure email. In *29th ACM User Interface Software and Technology Symposium (UIST)*, Tokyo, Japan, 2016. ACM.

[64] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. Seamons. Messageguard: A browser-based platform for usable, content-based encryption research. *arXiv preprint arXiv:1510.08943*, 2016.

[65] S. Ruoti, N. Kim, B. Burgon, T. Van Der Horst, and K. Seamons. Confused Johnny: when automatic encryption leads to confusion and mistakes. In *9th Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[66] M. D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *2014 ISOC Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2014.

[67] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *2nd Symposium On Usable Privacy and Security (SOUPS)*, 2006.

[68] J. Silver-Greenberg, M. Goldstein, and N. Perlroth. JPMorgan Chase hacking affects 76 million households. The New York Times, 2014. http://dealbook.nytimes.com/2014/10/02/jpmorgan-discovers-further-cyber-security-issues/. Accessed: September 23, 2015.

[69] K. Singh, H. J. Wang, A. Moshchuk, C. Jackson, and W. Lee. Practical end-to-end web content integrity. In *21st International World Wide Web Conference (WWW)*, pages 659–668, 2012.

[70] R. Sleevi and M. Watson. Web cryptography API. http://www.w3.org/TR/2014/CR-WebCryptoAPI-20141211/, 2014. Accessed: September 23, 2015.

[71] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. In *Financial Cryptography and Data Security*, pages 250–259. Springer, 2012.

[72] Symantec. Symantec Internet security threat report. http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf, 2008. Accessed: September 23, 2015.

[73] M. Ter Louw, K. T. Ganesh, and V. Venkatakrishnan. AdJail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *19th USENIX Security Symposium*, pages 371–388, 2010.

[74] C. Terhune. UCLA Health System data breach affects 4.5 million patients. Los Angeles Times, 2015. http://www.latimes.com/business/la-fi-ucla-medical-data-20150717-story.html. Accessed: September 23, 2015.

[75] S. Van Acker, P. De Ryck, L. Desmet, F. Piessens, and W. Joosen. WebJail: Least-privilege integration of third-party components in web mashups. In *27th Annual Computer Security Applications Conference (ACSAC)*, pages 307–316, 2011.

[76] E. Vaziripour, M. O'Neill, J. Wu, S. Heidbrink, K. Seamons, and D. Zappala. Social authentication for end-to-end encryption. In *2nd Workshop on "Who Are You?! Adventures in Authentication" (WAY) at the Symposium on Usable Privacy and Security*, 2016.

[77] Virtru. Virtru. https://www.virtru.com/. Accessed: September 20, 2015.

[78] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Technical Conference*, pages 321–334, 2008.

[79] M. West and D. Veditz. Content security policy. https://w3c.github.io/webappsec/specs/content-security-policy/, 2015. Accessed: September 23, 2015.

[80] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

[81] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang, et al. Schematizing trust in named data networking. In *2nd International Conference on Information-Centric Networking*, pages 177–186. ACM, 2015.

[82] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(3):66–73, July 2014.