Stickler: Defending against Malicious Content Distribution Networks in an Unmodified Browser

Amit Levy, Henry Corrigan-Gibbs, and Dan Boneh | Stanford University

Stickler guarantees the end-to-end authenticity of content served to users while simultaneously letting website publishers reap the enormous benefits and cost savings of content distribution networks without requiring browser modifications.

Many websites use content distribution networks (CDNs) to serve static assets such as images and JavaScript, but few realize that using a CDN potentially puts their users at risk. CDNs break the assumptions that Web security protocols rely on for data integrity. Transport Layer Security (TLS) is the industry standard for protecting data's confidentiality and authenticity while it's in transit on the Web. It secures the communication channel (a TCP connection) between two Internet nodes by encrypting and authenticating the bytes on that channel. Websites use TLS to ensure that the content in users' browsers matches what the publisher served. If a user connects directly to a website publisher's servers, then TLS is sufficient to authenticate the content served over the connection. However, when an intermediary, such as a CDN, is present between the user and the publisher, the TLS connection terminates at the CDN's servers. TLS ensures that the connection to the CDN is authenticated, but it says nothing about whether the CDN is serving the publisher's intended content. Essentially, the publisher and user must completely trust the CDN to faithfully serve the site's assets.

But how much should website publishers trust the CDNs that host their sites? Today, publishers have no choice but to assume that CDNs aren't modifying their sites' JavaScript, images, and other assets en route to users. However, this assumption isn't always reasonable. In the past few years, a small number of CDN providers have emerged to provide content delivery services to an increasing number of websites—CloudFlare alone claims to host content for more than 2 million websites (www.cloudflare.com/customers). These "consumer-grade" CDNs are much cheaper and have a much looser relationship with their customers than traditional CDNs. Typically, website publishers sign up for an account with low-cost CDNs via a click-through Web interface and often pay nothing for the service. Even though these CDNs have a tenuous business relationship with the sites they host, the sites' publishers are implicitly delegating a huge amount of trust to them: in at least one case, a no-cost CDN was able to

 Copublished by the IEEE Computer and Reliability Societies

generate Certificate Authority–signed TLS certificates for its publishers' domains without the publishers' intervention.[1] Today, website publishers must weigh the uncertain risks of using such a service against the undeniable benefits, including better availability and cheaper bandwidth.

Is it possible to get the best of both worlds? We argue that yes, website publishers can guarantee end-to-end integrity of their content while harnessing the benefits of third-party services such as CDNs. To do so, users' browsers must be able to authenticate website content integrity regardless of what serves it. Unfortunately, today's browsers don't provide a mechanism to authenticate content. To address this problem, we built, deployed, and evaluated a prototype system, Stickler, that lets website publishers guarantee the integrity of their content end to end, in the face of malicious CDNs, without modifying existing browsers or CDNs. Stickler separates connection authenticity from content authenticity by signing content directly with a private key that the site owner never has to share with the CDN. We found that connection authenticity (through TLS) is required only the first time a user visits a site, to bootstrap trust. Once trust is established, the user can fetch the website's content from any source, from a minimally trusted CDN to a peer-to-peer CDN such as CoralCDN.[2]

> **Website publishers must weigh the uncertain risks of using content distribution networks against their undeniable benefits, including cheaper bandwidth.**

CDN Benefits and Risks

CDN providers typically have edge-caching servers in multiple locations around the world so that their caches are geographically close to a site's end users. When a user makes a Web request, the CDN serves it from its cache or forwards the request to the publisher's server, caches the response, and returns the response to the user. A website publisher chooses to serve its content through a CDN for several reasons:

- CDNs can serve the bulk of a site's assets from a long-lived cache, dramatically reducing load on the publisher's servers.
- CDNs maintain edge servers around the world, so they can service cached content to clients with relatively low end-to-end latency.
- CDNs let publishers maintain availability in the face of a rapid traffic spike (a "flash crowd").[3]

CDN-cached assets are often served from a CDN-controlled domain (such as nytimes.procdn.biz) or a publisher-controlled subdomain (such as procdn.nytimes.com). Alternatively, a publisher might point its site's DNS records to the CDN's servers to let the CDN serve as a front-end proxy for all site requests. Either way, once the website has directed the user to retrieve content from a CDN, there's no way for the user to verify that content's authenticity.

Should website publishers be worried? We think they should, particularly if those publishers aren't paying for their CDN service. CDNs can inject ads to increase revenue, be compelled (for instance, by powerful governments) to modify JavaScript assets to leak passwords, or downsample image files to reduce their own bandwidth costs. Moreover, although publishers might be able to catch misbehaving CDNs by sampling CDN-served content, they might not catch CDNs that modify content in targeted ways (for example, only for certain Internet Protocol [IP] addresses, in certain countries, or after certain times).

Other researchers have identified this problem with CDNs. Jinjin Liang and his colleagues proposed letting publishers specify a whitelist of TLS certificates that the browser should trust to serve content in their DNS records,[4] thereby allowing publishers to revoke a CDN's certificate if they discover that the CDN is, for example, modifying site content. Peer-to-peer systems such as Firecoral have had to address a similar problem because the peers in them aren't trustworthy.[5] Unfortunately, such systems have required significant changes to browsers, which has prevented researchers from evaluating them in the wild at any scale.

An Updated Threat Model for the Web

Web browsers and website publishers should provide end-to-end integrity protection for content served to clients through intermediaries, including CDNs. In particular, a client should execute only JavaScript code signed by the publisher and load only publisher-signed content into the DOM (Document Object Model), even in the presence of active CDN attacks. We don't trust the CDN for integrity. In particular, we assume that it could

- serve stale content from the cache instead of fetching a newer version from the website,
- inject malicious client-side code into proxied responses (to sniff passwords or track clients),
- modify page content (to include advertisements),

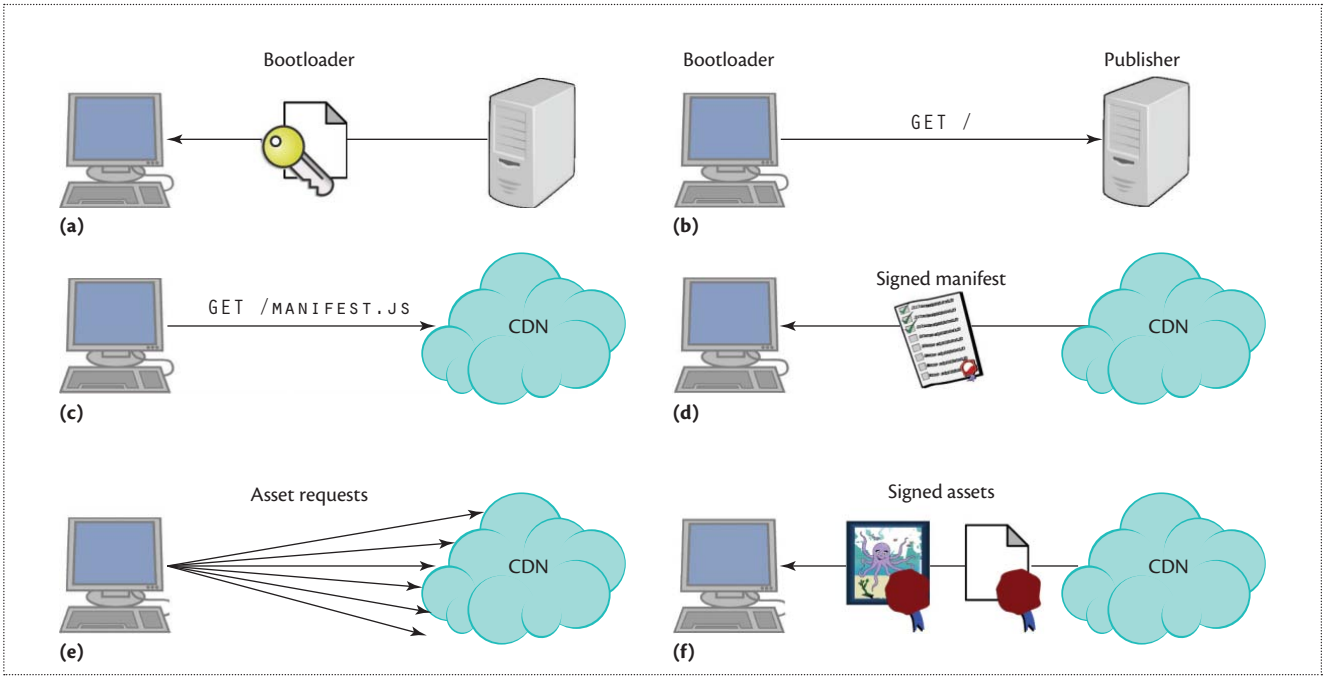

Figure 1. A bootstrap process executes when a client visits a Stickler-protected website. (a) The user's browser first fetches the site's index page. (b) The site server returns the Stickler bootloader code. (c) The bootloader script requests the manifest via XMLHttpRequest. (d) The content distribution network (CDN) serves the signed manifest file. (e) Executing the manifest instructs the user's browser to request the rest of the assets. (f) The CDN serves the signed site assets (images, videos, and so on) to the browser.

- downsample media files (to save bandwidth), and
- respond in arbitrarily malicious ways to client requests.

Providing strong integrity guarantees for CDN-hosted content is critical because integrity and confidentiality are closely linked. For example, if the CDN can insert JavaScript code into HTML pages served to Web clients, then it can read and exfiltrate passwords and other secret data via the client's DOM.

We do, however, trust the CDN for availability: a site's content might not remain available if a CDN refuses to serve that content. Because CDNs have a profit motive to maintain availability (although not necessarily integrity), we argue this model approximates the behavior of "consumer-grade" CDNs.

Stickler

Stickler is a prototype system that gives us a glimpse into how websites and users could securely communicate in today's Web, without trusting CDNs. Stickler doesn't require modifications to the browser or CDN—rather, it provides an architecture for structuring a website that separates data authenticity from connection authenticity (see Figure 1).

Design Overview

When a user browses to a Stickler-protected website, the DNS record for the requested domain points the browser to a webserver controlled by the publisher, not the CDN. The browser makes a TLS connection to this server and requests the website's index page.

The publisher's webserver returns an HTML page with the Stickler bootloader script embedded. This script contains the publisher's public signature verification key, JavaScript code to download and verify the site's assets, and the location of the site's manifest file. Because this first request is made over a TLS-authenticated connection directly to the publisher's server, authenticating the connection is sufficient to authenticate the content. The publisher never needs to share the private key it uses to authenticate this initial connection.

From this point on, the user can make all requests through the CDN, as all data is individually signed with the publisher's key, which is embedded in the bootloader. When the user's browser executes the bootloader script, the script initiates a request to the CDN for the site's manifest file. Upon receiving the site manifest, the bootloader script first checks that the manifest carries a valid signature by the publisher, and then the bootloader executes the manifest file as JavaScript. Executing the manifest file instructs the user's browser to generate a series of requests to the CDN for the rest of the site's assets, each of which bears a digital signature

from the publisher. When the CDN serves these assets, the bootloader verifies the publisher's signature on each asset and then processes it by invoking a function defined in the manifest. Typically, this processing just involves inserting the object into a prespecified location in the DOM. The user's interaction with the site (such as by clicking a link) could trigger more remote asset loads and signature verifications as needed to update the page content.

The publisher's server needs to serve only the initial bootloader script—the user can request all the site's other assets directly from the CDN—and the bootloader changes only if the secret key changes, so it can be cached on the user's browser. This means that even though requests must go directly to the publisher's server, it will only affect load times the first time a user visits the site and should have little impact on the publisher's overall bandwidth usage. In our implementation, the gzipped bootloader is 1.2 Kbytes, meaning that 1 million unique visitors would require only 1.2 Gbytes of bandwidth (currently less than $0.11 on Amazon EC2).

Content Authentication

When the Stickler bootloader downloads an asset from the CDN, the bootloader verifies it (to prevent the CDN from maliciously modifying it). An asset can either be signed with the publisher's private key, or a cryptographic hash of its contents can be embedded in the manifest; the publisher decides which to choose.

In general, a digital signature is useful if the content is likely to change since the manifest need not change. Moreover, multiple assets can exist concurrently, for example, if the CDN serves old (but still valid) cached assets. Embedding the hash is suitable for content that's unlikely to ever change or where doing so is difficult or infeasible. For example, using a cryptographic hash to verify an asset lets it reside on a server the publisher has no control over, for example, a JavaScript or Cascading Style Sheets (CSS) library served from a third party (such as Bootstrap or jQuery). In our performance evaluation, we found no significant client-side performance difference between using cryptographic hashes versus digital signatures for verification.

To facilitate content verification, Stickler requires a modification to the website authoring process. When the publisher builds its site, it must digitally sign each asset served to the client. The digital signing process can happen "on the fly" at page-load time for dynamic portions of the site's content and at compile time for static portions of the site's content. The requirement to sign assets isn't as burdensome as it might sound: many publishers already run their HTML, JavaScript, and CSS files through minifiers and compression tools as part of their "asset pipeline." Adding a digital signing phase to this pipeline would be relatively straightforward.

Dynamic Sites

A Stickler client-side application can use two different methods to access dynamic content.

The first is to fetch dynamic content directly from the publisher's server over TLS, bypassing the CDN entirely. Because the publisher's server is already hosted on a separate domain from the CDN's servers, the client can connect directly to this domain to download dynamic content. The bootloader need not verify the publisher's signature on assets fetched directly from the publisher's server. In this case, the remote peer and the content author are the same principal, so TLS connection-level authentication is sufficient.

The second method is to fetch dynamically generated assets via the CDN. This method might be useful if many clients will request the same dynamic asset and the CDN can cache the asset across client requests. In this case, the bootloader does need to verify the publisher's signature on the asset to prevent the CDN from tampering with the asset in transit. To allow for this integrity protection, the server-side application producing the dynamic asset simply bundles the asset as a digitally signed blob that the bootloader can decode. The publisher can implement this signing-and-packaging process as part of its dynamic asset generation pipeline. The downsides of this approach are that it requires the publisher's server to digitally sign every generated object and that it requires the publisher to store its secret key online, where it might be less secure.

Limitations

Stickler has several important drawbacks. First, it requires the publisher to sign every asset served to the client via a CDN. For some publishers, this might be relatively straightforward, but for sites with a very large number of preexisting assets, signing every single one might be infeasible.

Second, Stickler imposes a performance penalty on the user. When visiting a Stickler-protected site, the user's browser might have to perform a large number of signature verification operations. Moreover, the user has to make an extra round-trip to the CDN to get the manifest file. Although we found in our measurements that these overheads won't have a major impact on user experience, they could be unacceptable for some publishers.

Finally, Stickler prevents (by design) some of the value-added services that certain CDNs provide, such as minifying JavaScript, shrinking images, or performing other sorts of lossy compression on files without explicit publisher intervention. A publisher that heavily

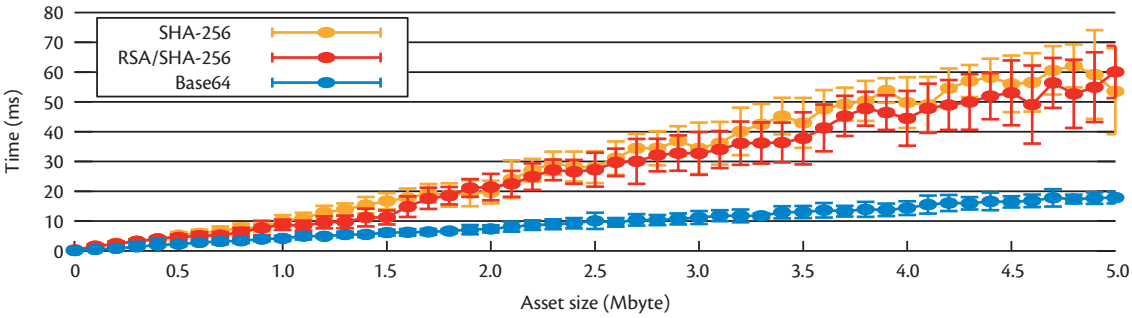

Figure 2. Performance. The top two lines represent the time to verify an asset using digital signatures (RSA/SHA-256) and cryptographic hashes (SHA-256) in Firefox depending on asset size. The difference in performance between the two methods isn't statistically significant: roughly 11 ms/Mbyte in asset size for each. The bottom line represents the additional time to encode the asset as a Base64 data-uri if it's an image or video. We ran the experiments on a 3.5-GHz quad-core Intel i7 running Firefox 35.

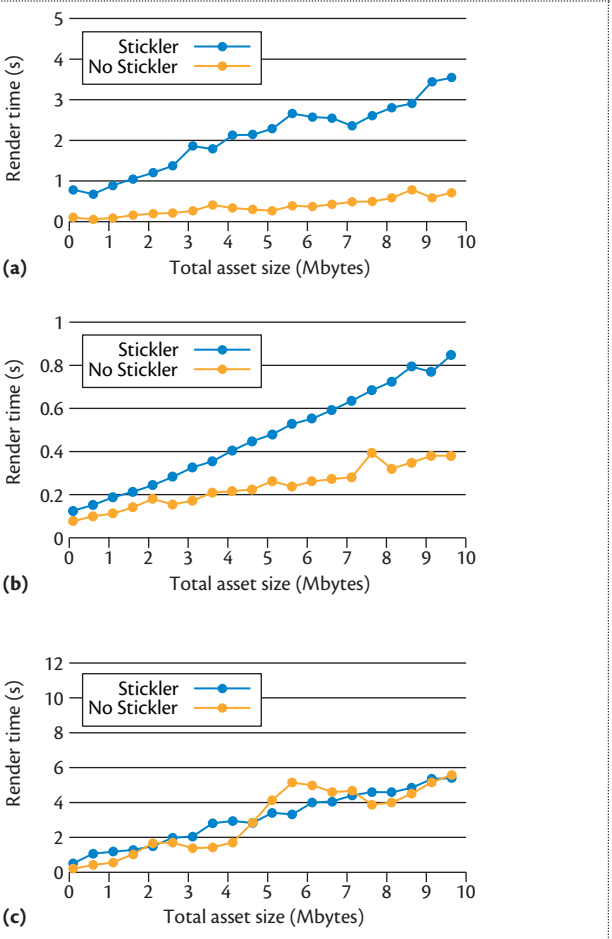

Figure 3. Overhead. Stickler imposes roughly (a) a 5× performance overhead on Firefox, (b) a 2× overhead on Chrome, and (c) negligible overhead on a mobile Firefox device. In these experiments, we loaded webpages into each browser while varying the number of images on the page. We measured the total time, including network latency to a server on the local host, to load the entire page. We ran the experiments on a quad-core Intel W3656 and a Google Nexus 5 mobile phone.

relies on its CDN to optimize media files and static assets values the benefit of this CDN-provided service over the risks of trusting the CDN for content integrity. However, it's important to note that Stickler doesn't prevent normal HTTP gzip compression.

Performance

Three factors govern performance after loading an asset with Stickler: network latency, cryptographic overhead (signature and hash operations), and the cost of loading the asset into the DOM (see Figure 2).

The overhead that Stickler imposes on network latency is small because file size increases only marginally. Stickler prepends each file, while in transit, with either a hash of the file contents or, if the asset is signed, a decryption of the hash. For example, using SHA-256 as the hash algorithm, Stickler adds 36 bytes to each file (32 bytes for the hash and 4 bytes for an expiration).

We found no significant performance difference between verifying a signature and a hash in the browser, which makes sense because signature verification is dominated by generating a hash of the contents. Loading assets into the DOM imposes an additional overhead in our prototype that we don't expect would manifest in a native implementation in the browser. The only way to insert binary assets such as images into the DOM from JavaScript is to encode them in Base64 first.

In practice, these overheads don't have a major impact on the user experience. We measured the end-to-end impact on performance of serving an app using Stickler in a controlled environment. We also deployed a Stickler version of the Stanford Applied Crypto Group website and have been measuring end-to-end performance for a large number of users.

Controlled Experiments

Figure 3 compares the time it takes to render a

Stickler-protected page with the time it takes to load an unprotected static HTML page as the number of assets on the page varies. In the experiment, we loaded a page with a varying number of 100-Kbytes image (between 1 and 96) over the local network. The experiments ran on an Intel W3565 quad core–based workstation with hyperthreading running Firefox 35 and Chrome 40, as well as on the Google Nexus 5 mobile phone running mobile Firefox 35.

As the figure demonstrates, the page render time increases roughly linearly with the number of images. Even with a relatively content-heavy page (10 Mbytes of media assets), the page renders within 1 second on the Chrome browser and within 6 seconds on a mobile phone running Firefox. Although the performance with and without Stickler on Chrome is comparable, Stickler imposes roughly a 5× performance penalty when using Firefox when the number of assets on the page is large.

Real Deployment

We deployed a Stickler version of the website for the Stanford Applied Crypto Group and instrumented it to measure the performance experience for site visitors, collecting data over a two-week period to verify that our local results were representative. We collected 617 total visits to the website—of those, roughly one-third contained resource requests that the cache couldn't serve. Figure 4 shows the total website load time as well as time spent verifying signatures. As in our local experiments, the data shows that signature verification constitutes a relatively small portion of total load time.

Figure 5 breaks down time spent on signature verification. As in our local experiments, verification time on Chromium-based browsers was minimal, but was more costly on Firefox. Whether this overhead is acceptable depends on the application's particular security requirements, but we expect that for especially sensitive sites (such as a health data site), publishers will willingly pay a performance cost for a security benefit.

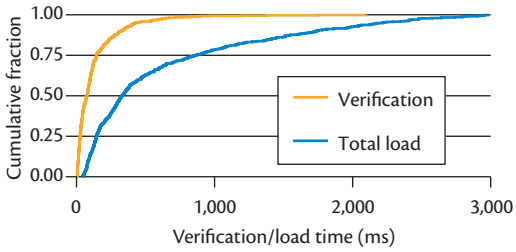

Figure 4. Website load time. Measurements from 617 visitors to the Stanford Applied Crypto Group's website confirms that signature verification constitutes a relatively small portion of total load time, whereas network performance dominates. At the 90th percentile, signature verification fell under 300 ms, whereas total load time was more than 3 s.

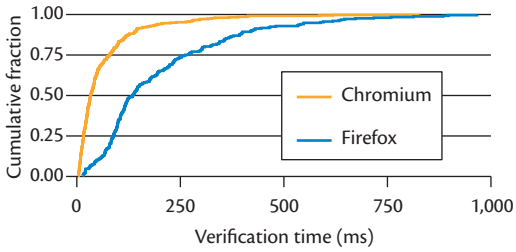

Figure 5. Time spent on signature verification. The 617 visitors to the Stanford Applied Crypto Group's website used Chromium- and Firefox-based user agents roughly equally (312 Chromium and 305 Firefox). Visitors using Chromium-based browsers were able to perform signature verification much faster (under 250 ms at the 90th percentile) than Firefox-based ones (approximately 750 ms at the 90th percentile). We didn't collect information about processor speeds, but our local experiments corroborate that this difference is, at least in part, due to performance differences between browser engines. We didn't include measurements from mobile clients in this figure.

Unlike the proposed W3C Subresource Integrity (SRI) mechanism for protecting CDN-served content's integrity, Stickler doesn't require browser modification. Although browser support for integrity protection would be ideal from a performance perspective, implementing and deploying these mechanisms in commodity browsers could take years and might never reach all browser vendors and platforms. For example, Server Name Indication, introduced in 2003 and widely used on servers to host multiple HTTPS sites on the same IP address, didn't gain widespread client adoption until 2011 and still isn't universally supported. Stickler is an effective short-term solution for browsers that will eventually support SRI and is a useful long-term solution for browsers and platforms (such as old smartphones) that might never support it. Our implementation and evaluation of Stickler demonstrate its practicality and performance. Website publishers can reap the manifold performance and cost benefits of using a CDN without having to put unnecessary trust in the CDN's correctness or honesty. ■

References

1. M. Prince, "Introducing Universal SSL," CloudFlare, 29 Sept. 2014; https://blog.cloudflare.com/introducing-universal-ssl.
2. M. Freedman, "Experiences with CoralCDN: A Five-Year Operational View," *Proc. 7th USENIX Conf. Networked Systems Design and Implementation* (NSDI 10), 2010, pp. 95–110.
3. P. Wendell and M.J. Freedman, "Going Viral: Flash

Crowds in an Open CDN," *Proc. ACM SIGCOMM Int'l Measurement Conf.* (IMC 11), 2011, pp. 549–558.
4. J. Liang et al., "When HTTPS Meets CDN: A Case of Authentication in Delegated Service," *Proc. IEEE Symp. Security and Privacy* (SP 14), 2014, pp. 67–82.
5. J. Terrace et al., "Bringing P2P to the Web: Security and Privacy in the Firecoral Network," *Proc. 8th Int'l Conf. Peer-to-Peer Systems* (IPTPS 09), 2009, p. 7.

Amit Levy is a PhD student in the Department of Computer Science at Stanford University. His work focuses on building pragmatic, secure systems for the Web that increase flexibility for application developers while preserving end-user control of private data. Contact him at levya@cs.stanford.edu.

Henry Corrigan-Gibbs is a PhD student in the Department of Computer Science at Stanford University. His research focuses on applied cryptography and computer security. Contact him at henrycg@stanford.edu.

Dan Boneh is a professor in the Department of Computer Science at Stanford University, where he heads the applied cryptography group. His research focuses on applications of cryptography to computer security. Boneh received a PhD in computer science from Princeton University. He has earned the Godel prize, the Packard Award, the Alfred P. Sloan Award, the RSA award in mathematics, five best paper awards, and the Ishii award for industry education innovation. Contact him at dabo@cs.stanford.edu.