SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements

Extended Version

Jeremy Clark and Paul C. van Oorschot
School of Computer Science
Carleton University, Canada
{clark,paulv}@scs.carleton.ca

***Abstract*—Internet users today depend daily on HTTPS for secure communication with sites they intend to visit. Over the years, many attacks on HTTPS and the certificate trust model it uses have been hypothesized, executed, and/or evolved. Meanwhile the number of browser-trusted (and thus, de facto, user-trusted) certificate authorities has proliferated, while the due diligence in baseline certificate issuance has declined. We survey and categorize prominent security issues with HTTPS and provide a systematic treatment of the history and on-going challenges, intending to provide context for future directions. We also provide a comparative evaluation of current proposals for enhancing the certificate infrastructure used in practice.**

***Keywords*-SSL; certificates; browser trust model; usability.**

I. Introductory Remarks

Enabling end users to easily communicate sensitive data online was a significant milestone in the development of today's web, and, arguably, a necessary condition for its explosive growth. Little-changed since its early days (1994–2000), the core SSL/TLS technology persists as the basis for securing many aspects of today's Internet including software download, data transfer, user passwords, and for site authentication. While centred on the HTTPS protocol (HTTP over SSL/TLS), its security services—confidentiality, message integrity, and site authentication—fundamentally rely on the correct interplay of out-of-band infrastructures, procedures, and trust decisions.

While the web has moved from serving static information pages to one which is relied on for billions of dollars of commerce and for supporting critical infrastructures, there has been an erosion of confidence in the HTTPS certificate infrastructure for multiple reasons, *e.g.,* increasing issuance of server certificates through fully-automated (domain validated) procedures, a proliferation of certificate authorities (CAs) which may either directly issue site certificates or certificates for other CAs, and the compromise of real-world CAs leading to increased concern amongst security experts of real-world man-in-the-middle (MITM) attacks on HTTPS.

SSL/TLS has evolved in response to the discovery of cryptographic weaknesses and protocol design flaws. Problems with the certificate model appear to be more challenging, including among others: design and implementation issues in the CA/Browser (CA/B) trust model leading to fragility (compromise of a single CA can, at least temporarily, undermine system-wide security) and lack of trust agility, poor support for certificate revocation, a reduction in CA diligence in certificate issuance, and user interface challenges related to reliably signalling to end-users, in ways not ignored or spoofed, security indicators and site authentication information.

In this paper, we provide a broad perspective of the SSL/TLS (henceforth TLS) mechanism, as employed with web browsers for securing HTTP traffic. We consider HTTPS, the underlying CA infrastructure, CA/B trust model, and proposed enhancements. Among many important HTTPS-related topics beyond our main focus are: phishing, performance enhancements, use of certificates for client-authentication, and the use of TLS beyond securing HTTP.

Our main contributions are the following: (1) We classify and put into a broader context disparate contributions on HTTPS security, spanning elements of cryptographic design and implementation, systems software, operations, and human factors. (2) We provide a comparative evaluation of existing proposals to enhance security aspects of the CA/B model, deconstructing and evaluating their core ideas. (3) Building on this contextual review, classification, and analysis, we summarize open problems and future research directions. In addition, by systematic discussion of security issues in a single place, we hope to provide perspective based on the hindsight of a multitude of historical problems. Our work highlights the overall complexity, including algorithms, protocols, infrastructure, configuration, and interfaces, and contributes an overall understanding of which issues are addressed by which enhancements and protocol revisions.

II. Background

Historical Objectives: SSL was developed to address Netscape's needs for securing web traffic, and specifically designed to work well with HTTP [85]. Network protection of data like credit card details sent from client to server motivated two major design goals: confidentiality, and server authentication—sensitive data should be released only to a party one would 'intend to do business with,' *i.e.,* the

Version: March 7, 2013. An extended abstract of this paper appeared at *IEEE Symposium on Security and Privacy* 2013 [33].

correct web server.[1] Client authentication was an optional third goal, however the credit card number largely replaced user identity. Even today, while TLS supports client-side authentication, this feature is little-used on the public internet; we do not consider it at length herein.

As Netscape intended SSL to be a core technology beyond use with HTTP alone, and since most high-runner internet protocols ran over TCP, SSL was designed to provide a general channel that can be adopted with minimal modification by almost any TCP-based protocol seeking some security. An important property was termed transparency: "the data that one end writes is exactly what the other end reads [85]."

Protocol Specification: HTTPS combines the network protocol HTTP with the cryptographic protocol TLS. The TLS protocol (v1.0, 1.1, 1.2) updates the older public SSL protocol (v3.0). TLS provides a secure tunnel to a server, which is most commonly authenticated by an X.509 certificate. Specification of the cryptographic primitives used by X.509 is largely delegated to PKCS standards. We do not focus on protocols (*e.g.,* IMAP or SMTP) other than HTTP run over TLS, nor the use of TLS with transport layer protocols other than TCP (*e.g.,* DTLS).

III. Crypto Protocol Issues in HTTPS

In this section, we consider attacks on the TLS protocol which relate to HTTPS security. Section IV expands focus to the broader CA/B infrastructure and human decisions involved. As TLS is well-documented, we assume familiarity with the basic protocol. Many attacks refine known techniques; examining both historical and recent attacks provides a fuller perspective.

A. Weaknesses in Cryptographic Primitives

1) Weak Encryption & Signature Key Lengths: Several encryption functions offered in the ciphersuites of early versions of TLS are no longer considered secure. Any symmetric key encryption scheme with 40, 56, or 64 bit keys is subject to a brute-force attack. TLS supported DES, RC2, and RC4 with some of these key lengths. Asymmetric encryption schemes like RSA are subject to factoring attacks when used with a 512 bit modulus. A 2007 analysis of TLS servers found that while only 4% of sites still offered RSA-512, 93% supported (single) DES [69]. Note that supporting an insecure primitive does not imply it is ever used, as security parameters are negotiated (but see Downgrade Attacks below). NIST strongly recommends that primitives hold the equivalent of 112 bits (symmetric) security strength and will require this by 2014 [23] (*e.g.,* phasing out 1024-bit RSA/DSA and 193-bit ECDSA).

Key length is also an issue for certificates. Sufficient key lengths should be used by the certificate authority to sign a certificate, and CAs should only sign certificates containing public keys that are of sufficient length.[2]

2) Weak Hash Functions: To issue a site certificate, CAs sign its hash. Collision-resistance of the hash is paramount: an adversary that could construct two meaningful certificates with the same digest could transfer a CA signature from a benign site certificate to a malicious CA certificate. The MD5 hash function, published in 1992, has been eligible for providing certificate digests. However the collision resistance of MD5 has deteriorated over time, from generic attacks [103] to the first published collision [38] to the generation of "meaningful" collisions [99], and finally finding collisions that are structured enough to be both an acceptable benign site certificate and a malicious root certificate [100]. Use of MD5 is discouraged (RFC 3279) and certificates digested with MD5 are in decline [55]. MD5 remains recommended in other places in the TLS protocol where collision-resistance of the hash function is not critical, *i.e.,* HMAC and key derivation [66], [26], [67].

B. Implementation Flaws and Related Attacks

1) PRNG Seeding: Many values in the TLS protocol are generated randomly, including secret keys. This requires a strong pseudorandom number generator (PRNG), seeded with a high entropy seed. The Netscape browser (prior to 1.22) relied on a PRNG implementation with weak keys [52] allowing the TLS session key (master secret) to be predictable. A 2008 change to the Debian operating system reduced the randomness served to OpenSSL, which was used to generate TLS certificates with predictable private keys [25], [16], [111]. Recently, 0.5% of TLS certificates were found to have recoverable RSA private keys due to shared prime factors [54], [71]; most originated from poor PRG seeding in embedded devices.

2) Remote Timing Attacks: Remote timing attacks have been used against TLS servers that use an optimized variant of RSA decryption, the default in OpenSSL versions prior to 0.9.7b [31], [14]. The decryption algorithm makes branching decisions that are functionally dependent on the long-term certified secret key. This results in measurable differences in execution time, leaking information about the key during TLS handshakes. Previous OpenSSL implementations of ECDSA enabled similar remote timing attacks [30].

C. Oracle Attacks

The following attacks interactively and adaptively query the victim's protocol implementation, treating it as an oracle.

1) RSA Encoding: SSL 3.0 with the RSA ciphersuite uses "textbook" RSA (which enables ciphertext malleability) for transporting a PKCS#1 v1.5 encoded premaster secret to the server during the handshake. If upon decryption and decoding, the plaintext is not properly encoded, an error

[1]The meaning of 'correct' remains challenging today (see Section VI).

[2]An intermediate CA in Nov 2011 was revoked for issuing certificates for 512-bit RSA keys. http://www.entrust.net/advisories/malaysia.htm.

is returned to the client. An adversary could capture an encrypted premaster secret, and, in separate handshakes with the same server, submit adaptively modified versions of it, learning if they are conformant [28]. With just this information, the adversary can eventually ($\sim$1M queries) recover the premaster secret. TLS 1.0 consequently recommends that encoding errors are handled indistinguishably from successful decryptions.[3]

2) CBC Initialization: In TLS 1.0 and earlier, all block ciphers are used in cipher block chaining (CBC) mode. Records are encrypted individually, however the initialization vector for each (except the first) is set equal to the last block of ciphertext sent (*i.e.,* in a predictable way). CBC with predictable IVs is not secure against chosen plaintext attacks [24], and thus an adversary capable of injecting partial plaintext into a TLS connection and of observing the transmitted ciphertext can determine semantic information about the rest of the plaintext [21], [22]. In one instantiation of this attack, BEAST, an adversary submits adaptively chosen cross-site requests for a domain with a secure cookie to learn the value of the cookie (and by adjusting the amount of the value included in a single block, due to partitioning, the value can be guessed byte-by-byte) [39]. This issue is resolved in TLS 1.1, not applicable to any stream cipher (*e.g.,* RC4), and is purportedly mitigated by first sending the first byte as a separate record ('1/n-1 record splitting').[4]

3) Compression: The use of data compression is a negotiable option in TLS, although one never broadly supported by browsers. TLS does not obfuscate the length of a compressed TLS record, thus again an adversary capable of injecting partial plaintext into a TLS connection and observing the post-compression record length can determine semantic information about the rest of the plaintext [65]. An instantiation of this attack, CRIME, used a similar setup to BEAST for recovering secret values from secure cookies [90]. As a result, all major browsers have disabled TLS compression.

4) CBC Padding: With CBC, TLS records are authenticated by appending a MAC, then padded with a variable-length padding scheme, and then encrypted. The padding scheme encodes the number of padding bytes to be discarded upon decryption. At a high level, the CBC padding attack [104], [32] is to capture a TLS record, modify it in a structured way (byte-by-byte, starting with the last byte, as facilitated by the CBC mode of operation), and submit the modified record to the server for decryption. If the ciphertext is rejected without a padding error (but, overwhelmingly, with a MAC error), the plaintext value of a targeted byte can be determined.

Error values are returned over TLS so even if the error messages are different for bad padding and bad MACs, the adversary cannot tell which error was returned. However it has been shown that the adversary can use a timing attack to distinguish them [32], as padding is efficiently checked while verifying the MAC requires processing each byte of the plaintext. As a result, major implementations invariantly compute the MAC, even with a padding error. However, with a padding error, it is not always possible to determine the length of the original message, and it was recently shown that even smaller timing differences in how much of the message has the MAC applied to it (since corrected in major implementations) could be used to distinguish a padding error [17]. When any decryption error occurs, the session is aborted. Thus the adversary must capture the same plaintext (or a plaintext with the secret in the same place, such as a cookie) encrypted under the new session key to continue the attack.

D. Protocol-level Attacks

1) Ciphersuite Downgrade Attack: The ciphersuite used by the client and server is negotiated during the TLS handshake. In SSL 2.0, a man-in-the-middle could influence the negotiation and downgrade the strength of the ciphersuite to the weakest acceptable by both parties. This is fixed in SSL 3.0 and all versions of TLS by having the client send, once the MAC keys have been established, an authenticated digest of the previous handshake messages and waiting for an authenticated confirmation from the server. Thus, downgrade prevention is contingent on the unavailability of weak MAC functions for negotiation.

2) Version Downgrade Attack: The TLS version is also negotiated and while version downgrade attacks are not possible against a strict implementation of the TLS specification, many client implementations respond to certain server errors by reconnecting with an older TLS version. These server errors can be spoofed by an attacker. To prevent an adversary from first downgrading to SSL 2.0 and then downgrading the ciphersuite, TLS prohibits downgrading to SSL 2.0. TLS implementations may still be vulnerable to downgrades from later version to earlier versions (*e.g.,* from TLS 1.1+ to TLS 1.0 to exploit CBC initialization vulnerabilities). One mitigation is to include the highest supported version number in the list of ciphersuites during negotiation, extending ciphersuite-downgrade protection to versions [5].

3) Renegotiation Attack: Once a TLS connection has been established, either party can at any point request a new handshake, within the existing tunnel, to renegotiate the cipher suite, session key, or other relevant connection parameters. The renegotiation protocol was discovered to be flawed in 2009[5] and was subsequently updated [1]. The erroneous version allowed an adversary to establish a connection

[3]Also note that while RSA key transport support is ubiquitous, 60–70% of servers also support Diffe-Hellmen key exchange [87], [105] which has the added benefit of perfect forward secrecy.

[4]A. Langley, "BEAST Followup," *ImperialViolet* (blog), 15 Jan 2012.

[5]M. Ray, "Authentication Gap in TLS Renegotiation," *Extended Subset* (blog), 4 Nov 2009.

to a server, send data, renegotiate, and pass the renegotiated connection onto a client that believes it is forming an initial connection. This effectively allowed the adversary to prepend chosen records to new HTTPS connections. An extension [51] to the standardized countermeasures [1] can provide a strong notion of renegotiation security.

4) Cross-Protocol Attacks: During a TLS handshake, the server sends a signed set of parameters for either a Diffie-Hellmen key exchange or RSA key transport as negotiated earlier in the handshake, however it does not explicitly state which key agreement algorithm is used. An adversary could request a set of signed Diffie-Hellmen parameters from a server that a client intends to use RSA key transport with, and replay them to the client [106]. It was proposed that the client would use the Diffie-Hellman parameters to 'encrypt' the session key using the RSA encryption algorithm, and the key could be easily recovered.[6] In practice, implementations do not parse the responses in a susceptible way. However, recently, the attack was successfully demonstrated on replaying ECDH parameters to clients expecting ordinary DH [75]. The attack can be mitigated by labelling all parameters with the algorithm they correspond to [10].

IV. Trust Model Issues in HTTPS

Section III narrowly considered attacks on the TLS protocol and the cryptographic algorithms it involves. This section assumes a perfectly functioning TLS protocol and considers attacks on the broader CA/B infrastructure. Our analysis covers the certification process itself, who is allowed to be a certificate authority (anchoring trust), how this authority can be delegated (transitivity of trust), how certificates are revoked (maintenance of trust), and how users interact with certificate information (indication and interpretation of trust). In what follows, we specifically note which issues remain unresolved.

A. Certification

A web certificate binds a public signing key to an 'identity.' The correctness of the binding is asserted through a digital signature, by a CA implicitly expected to maintain the accuracy of the binding over time. TLS enables client software to establish a confidential channel terminated by the entity holding the private key associated with the certificate.

The essential attribute that all HTTPS server certificates have is a domain name which the certificate holder controls. This is placed in the `commonName` (`CN`) attribute under `Subject`, unless one or more domains are indicated in the subject alternative name field in an X.509 extension. If an entity requests a certificate for a domain name, the CA will typically challenge the requester to demonstrate control over the domain. Note that this implicitly assumes that domain names are mapped to the correct webserver (IP address), a mapping accomplished through DNS. Such certificates are called domain validated (DV) certificates.

Issued certificates may include additional CA-verified information, such as organization name and postal address. Validation procedures have degraded over time, exemplified by more CAs using a completely automated process (*e.g.,* automated DV certificates). In response, the CA/Browser Forum established extended validation (EV) certificates and guidelines for their issuance,[7] including diligent human validation of a site's identity and business registration details.

Security Issues (Certification)

Hostname Validation (CAs): Automated domain validation services provided by a CA will typically send a validation email to a fixed email address associated with the `CN`'s top-level domain (*e.g.,* `admin@domain`) or one taken from `CN`'s WhoIS record. Both mechanisms rely on accurate domain information; thus any disruption to the CA's ability to receive accurate DNS records (*e.g.,* DNS cache poisoning [62], [95]) could result in an improperly issued certificate. For email validation, CAs should also ensure the email address is only accessible to the site administrator. For example, a certificate for the login page of Microsoft's public webmail service, `login.live.com` (formally Hotmail), was wrongfully issued by a CA that offered to validate through `sslcertificates@live.com`, an email address that was open to public registration [112].

Even with non-automated validation, an adversary may employ social engineering. For example, in 2001, a CA wrongfully issued two certificates to someone posing as a current Microsoft employee.[8]

Hostname Validation (Clients): Although current browser platforms validate that a received site certificate matches the hostname, some non-browser software had inadequate validation. Many mobile applications display HTTPS content and one study found that 1074 of 13500 Android apps did not validate the hostname (aside from other TLS implementation flaws) [43]. A concurrent study identified the lack of hostname validation in cloud clients (Amazon's EC2 libraries), e-commerce backend systems (Paypal's SDK), online shopping carts, ad networks (AdMob), and other non-browser software employing HTTPS [50].

Parsing attacks: Flaws relating to parsing enable improper issuance (incorrect CA parsing) and validation (incorrect browser parsing) of certificates. Certificate requests containing a null character (Ø) in the `CN` can be misinterpreted. For example, a `CN` of `bank.comØevil.com` was validated by some CAs' automated domain validation as

[6]More specifically, the attack assumes the prime group order and generator in DH are respectively used as an RSA modulus (which should be the product of two primes) and public key exponent. Plaintext recovery reduces to finding roots in prime ordered groups.

[7]CA/Browser Forum: Guidelines For The Issuance And Management Of Extended Validation Certificates (v1.4), 2012.

[8]Microsoft MS01-017: Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard, 22 Mar 2001.

`evil.com` while browsers have been known to accept it as a valid `CN` for `bank.com` [63], [72]. A dangerous variation is `*Øevil.com`, which grants a universal wildcard certificate acceptable to older NSS-based browsers [72].

Some CAs and browsers have also inconsistently interpreted the object IDs specifying which string is the `commonName`: for example, `CN` is identified by OID 2.5.4.3 but some browser parsers accepted 2.5.4.003 or 2.5.4.18446744073709551619 (64-bit uint overflow) as the `CN`, while some CAs ignore them [63].

EV downgrading: Many of the problems associated with automated domain validation are claimed to be thwarted by EV certificates. However a site that holds an EV certificate can be downgraded to normal HTTPS by a man-in-the-middle (MITM) attack with a fraudulent DV certificate. Furthermore, such an adversary can arrange for the EV certificate to be displayed through a "rebinding" attack [113], [97] that is consistent with the browsers' origin policy [56].

B. Anchoring Trust

Validating that a certificate request comes from the entity specified in the `SubjectName` is an important CA function. As no one entity has universal control of all namespaces, it is not clear who is best suited for such validation. As a result, there exists a spectrum of CAs, with the majority of site certificates being issued by commercial CAs with ties to the security or domain registration industries.

The CAs responsible for issuing the most site certificates, and their industry sectors, are: Symantec (anti-virus), Comodo (security tools), Go Daddy (domain registrar), GlobalSign (primarily a CA), DigiCert (primarily a CA), web.com (web services), and Entrust (telecommunications). Note Symantec includes Verisign (computer security), GeoTrust (primarily a CA) and Thawte (secure servers). Following these is a long tail of CAs, each responsible for less than 2% of site certificates observed on the web.

Software vendors (*e.g.,* Microsoft, Apple, Mozilla, Opera) configure a default list of self-signed CA certificates in operating systems and/or browser as *trust anchors*. Each HTTPS site whose site certificate the browser accepts is thus de facto trusted by users because its certificate has been vouched for (directly or indirectly) by at least one of the trust anchors. Mozilla's Firefox 15, for example, includes $\sim$150 trust anchors from $\sim$50 organizations. However since CAs with trust anchors can issue certificates empowering other organizations to act as a CA (see below), the number of automatically trusted CAs is much larger. The SSL Observatory reports that between Microsoft's Internet Explorer and Mozilla's Firefox, $\sim$1500 CA certificates from $\sim$650 organizations[9] in $\sim$50 countries are browser-accepted [40].

[9]The reported number of organizations may be inflated due to variation in organization name (or division) across certificates. Also in some cases, the issuing CA retains actual possession of the intermediate certificate.

On private networks, particularly in corporate environments, a root certificate for the organization may be configured as a trust anchor on employees' machines. The organization can then proxy (*i.e.,* MITM) HTTPS connections with middleware boxes specifically designed for this task to perform content inspection. The corporation may even be able to obtain a browser-accepted CA certificate for doing this, although issuing such a certificate is against CA policies. For example, Trustwave admitted to issuing certificates for this purpose but later revoked them,[10] while, purportedly by accident, TURKTRUST issued certificates that were discovered being used this way.[11] The mobile browser OperaMini openly proxies HTTPS connections in this way to allow compression between the client and proxy [84]. Proxies assume all responsibility for certificate validation, with a recent study finding that many implementations had validation flaws [60].

Security Issues (Anchoring Trust)

CA Compromise: Without further enhancement (*i.e.,* without the primitives evaluated in Section V), any trusted CA can issue a browser-acceptable certificate for any site. Thus an adversary can target the weakest CA to obtain a fraudulent certificate and, assuming clients would not notice a different CA, this certificate enables the adversary to evade detection in a MITM attack. In 2011, two CAs were compromised: Comodo[12] and DigiNotar.[13] In both cases, certificates for high profile sites were illegitimately obtained and, in the second case, reportedly used in a MITM attack.[14]

Compelled Certificates: Concerns have also been raised about the abilities of nation-states to compel certificates from a browser-accepted CA [94]. Governmental entities are often well-positioned to proxy (*i.e.,* MITM) HTTPS connections by controlling network infrastructure and/or compelling ISPs. For example, reportedly, HTTPS connections to Facebook over multiple ISPs within Syria were MITMed with a Facebook certificate issued by the Syrian Telecom Ministry.[15] In this case, the Ministry was not a browser-acceptable CA, and so it is not an example of a state compelled certificate, but one that demonstrates the danger posed.

C. Transitivity of Trust

Given that trust anchors can issue intermediate CA certificates (and intermediates can be enabled to do the same), a

[10]Bug 724929, Bugzilla@Mozilla, reported: 7 Feb 2012.

[11]A. Langley, "Enhancing digital certificate security," *Google Online Security Blog*, 3 Jan 2013.

[12]J. Appelbaum, "Detecting Certificate Authority compromises and web browser collusion," *Tor Blog*, 22 Mar 2011.

[13]"Black Tulip Report of the investigation into the DigiNotar Certificate Authority breach," *Fox-IT* (Tech. Report), 13 Aug 2012.

[14]C. Arthur, "Rogue web certificate could have been used to attack Iran dissidents," *The Guardian*, 30 Aug 2011.

[15]P. Eckersley, "A Syrian Man-In-The-Middle Attack against Facebook," *EFF Deeplinks* (blog), 5 May 2011.

site certificate is browser-acceptable if the browser can build a chain of certificates that lead to a trust anchor. One study found 20% of valid certificates required no intermediate and 38% used one [19].

The path validation algorithm is specified in RFC 5280 to begin with the server certificate and build the path "forward" to the trust anchor, although there are efficiencies to building in reverse [41]. Certificate chains are subject to constraints. Intermediate CA certificates must be authorized to be a CA (`CA:TRUE` under `basicConstraints`). CA certificates may also restrict the number of CAs that can precede it (*i.e.,* toward the leaf) in the chain (*e.g.,* `pathlen:0` under `basicConstraints` means the CA cannot delegate further CAs and can only sign leaf certificates).

Servers are mandated to present an entire chain, but in practice, browsers may use a chain discovery mechanism (*e.g.,* AIA: Authority Information Access). Intermediate CAs are invisible to client software until their certificate is encountered, and yet they are essentially as trusted as an anchor. This makes it difficult for users or OSs/browsers to preemptively know about and remove unacceptable intermediate CA certificates. As above, while only $\sim$50 organizations have visible trust anchors, many more organizations have acceptable intermediate CA certificates. Technically, Ford, Marks and Spencer, and the US Dept. of Homeland Security are authorized for issuing acceptable certificates for any website [40].

Security Issues (Transitivity of Trust)

Basic Constraints: Certificate path validation must also check the constraints during validation, in particular that each intermediate CA certificate has `CA:TRUE` set under `basicConstraints`. If this is not checked, a certificate obtained for a webserver could issue browser-acceptable certificates for any other website. Initially not checked by Microsoft's CryptoAPI,[16] this has now been patched.[17] A decade later, the issue resurfaced in Apple's iOS.[18]

D. Maintenance of Trust

Another important function of a CA is to terminate the validity of a certificate prior to its preconfigured expiration date upon becoming aware of certain circumstances, *e.g.,* mistaken issuance, site or CA compromise, affiliation change, a superseding certificate, or cessation of the holder's operations [46]. Revocation status must also be available through the issuing CA, *i.e.,* by certificate revocation lists (CRLs) or online certificate status checking protocol (OCSP) responders. CRLs are signed by the CA's key while OCSP responses are produced by servers designated by the CA. CAs often prefer OCSP responders as they can be updated on-demand without use of the (generally offline) CA signing key and due to response size.

In practice, some CA certificates do not include any revocation information, and when OCSP responders are specified, they are often unresponsive. Thus, current browsers fail open, accepting certificates for which revocation information cannot be located (browsers should downgrade all EV certificates to a regular certificate, or warn, as responsive revocation is an EV requirement).[19] In response to the failings of revocation, some browsers (*e.g.,* Chrome) maintain an updatable certificate blacklist (see Section V-C).

While the mandatory expiration date field provides an eventual default form of revocation, many certificates are valid for multiple years (the median lifetime varies across measurements: 12 [87], 12–15 [55], or 24 [19] months).

Security Issues (Maintenance of Trust)

Blocking Revocation: If an adversary is able to obtain a fraudulent certificate for a site which is subsequently revoked, it may take several days for this information to be available to clients, even with OCSP, due to caching [102]. Even then, the clients may not be able to reach an OCSP responder or CRL distribution point. Further, a MITM adversary could respond to a client's request with an HTTP error (*e.g.,* error 500: internal server error) or OCSP error (response status 3: try again later [72]); in this case, the revoked certificate typically continues to be accepted.

Similarly, a MITM adversary could prevent a browser blacklist from updating but this would require persistent blocking—the OCSP check would only occur when the client encounters the fraudulent certificate and the MITM adversary is already in position. The CA's CRL could have been obtained by checking an unrelated certificate.

Ownership Transfer: Since TLS site certificates are bound to a domain name, certificates should be revoked when domain ownership expires or is transferred. This is however not typically enforced. For example, Facebook (the target of the Syrian MITM attack mentioned above) acquired fb.com for $8.5M in 2010 but can have no assurance that the previous owner does not have a valid unexpired certificate for the site that could enable a MITM attack.[20]

E. Indication and Interpretation of Trust

Some HTTPS security protections rely on user due diligence. Perhaps naively, users are expected to verify the outcome of each connection attempt, typically indicated by a visual cue in the browser window. More diligent users may verify certificate details; *e.g.,* that the subject name—organization, address, country—matches their expectation.

[16]M. Marlinspike, "Internet Explorer SSL Vulnerability," *thoughtcrime* (online), 5 Apr 2002.

[17]MS02-050: Microsoft certificate validation flaw, 2002.

[18]CVE-2011-0228: iOS certificate chain validation issue in handling of X.509 certificates, 2011.

[19]A. Langley, "Revocation doesn't work," *ImperialViolet* (blog), 18 Mar 2011.

[20]This example is meant to illustrate the principle rather than a vulnerability with Facebook. Few users type in https://fb.com.

Finally, the browser may require users to respond to warning dialogues in some cases.

Browser Security Cues: Desktop browsers typically use two primary cues to indicate a website is being accessed over HTTPS: (1) the URL in the address bar begins with `https://` and (2) a lock icon is displayed somewhere in the browser's chrome (*i.e.,* the boundary region of the window populated by the browser itself). Typically, clicking on the lock icon will display information about the certificate. One impedance to better user understanding of browser security indicators is the inconsistency of how cues are implemented across browsers [20]. Guidelines for browser cues have been published.[21]

One study used eyetracking to find that of 16 primed participants interacting with an HTTPS site, 11 viewed the lock, 7 the `https://` indicator, and only 2 interacted with the lock to display certificate information [109]. Another study found that 63 of 67 participants logged into a hypothetical banking website with all HTTPS indicators removed, suggesting that many did not notice the difference [91] (although one researcher argues this is enlarged by flaws in the study[22]). The introduction of EV certificates added a new desktop browser cue, typically inducing the colour green in the address bar itself or its font. Today's desktop browsers also display the organization name from the certificate in the address bar, alongside the URL. One study found that of 28 users, the "less than 40%" who actually looked at browser cues (measured via eyetracking) was slightly but not significantly more likely to interact with an EV site than a DV (domain validated) site [93]. Aside from cues, some users appear to simply assume a page is secure based on the information being requested [47], [36].

Browser Security Warnings: Browsers display warnings if an HTTPS connection fails for certain reasons. One study found that 30 of 57 users clicked through and logged into a simulated banking website when it displayed a certificate warning page [91]. Another study asked users about three common warnings: expired certificates, certificate chains not terminating in a trust anchor, and certificates not identically matching the visited site's domain. Of 409 users, only 36% understood the expired certificate warning, 28% the unknown CA warning, and 40% the mismatched domain warning [101]. Of the users who did not understand, just over a third claimed they would ignore each.[23] Of the users who did understand, 63% would ignore the expired certificate warning, 33% the unknown CA warning, and 17% the mismatched domain. The study authors reworded the warnings and saw improvements in understandability but felt users still opened themselves to MITM attacks in their actions. This study was replicated; users were found to ignore warnings more than they self-report, and no measurable difference was observed between ignoring rewritten warnings and the browser defaults [96].

Mixed Content: A prevalent warning not considered in the above studies is mixed content—the ability to weave dynamic content from multiple sources into a single website. When a site is accessed over HTTPS, browsers will issue a mixed scripting warning if the site embeds any scripting resources (*e.g.,* Javascript, CSS, or even SWF [56]) that are not accessed over HTTPS. This is important because the script runs with the HTTPS site's privileges. Some browsers (*e.g.,* Chrome) follow the lead of the Gazelle research browser [107] and actively block insecurely loaded scripts. Non-scripting resources (*e.g.,* static images) or resources loaded into an iframe will typically generate a less severe mixed content warning (*e.g.,* an `https://` cue but no lock or a lock with a caution sign; clicking on the cue displays the warning). Finally, if an EV certified site embeds DV certified scripting, the EV cues are still displayed, which can be exploited (see Section IV-A above) [113], [97].

Mobile Browsers: With the advent of smartphones and tablets, users also access HTTPS sites on mobile browsers. Mobile browsers conform less to HTTPS user interface guidelines than desktop browsers, with less support for displaying certificate/connection details, distinguishing EV certificates, or warning users about mixed content [20].

HTTPS Form Submit: A relatively common practice is including a login box on an HTTP page, but arranging any login information to be submitted over HTTPS.[24] This allows the overhead of obtaining the certificate and establishing a TLS connection to be avoided for users who visit the page without logging in. Browsers include a general warning (at least the first time) when any information is not submitted over HTTPS but since many submissions do not require security (*e.g.,* comments, posts, feedback, *etc.*), it is sensible for users to disable this warning (an option provided prominently). Beyond this, a user is given no cue that sensitive information will be transmitted over HTTP or HTTPS, let alone to which hostname. Of the Alexa top sites with login pages, 19 of 125 offered such a post-to-HTTPS login page while 56 of 125 only used HTTP [98].

Security Issues (Indication and Interpretation of Trust)

Stripping TLS: Given users' inattentiveness to HTTPS security indicators and warnings, a MITM adversary may thwart HTTPS in a technically detectable manner but one unlikely to be noticed. For users being redirected to an HTTPS site, arguably the most astute attack is to simply relay pages back to the user over HTTP—an attack now called SSL stripping after the sslstrip tool [72], although it had been noted earlier [80]. On login pages served over

[21]Web Security Context: User Interface Guidelines. *W3C Recommendation*, 12 August 2010.

[22]A. Patrick. "Commentary on Research on New Security Indicators," (online), 6 Mar 2007.

[23]If they were hypothetically warned while at Craigslist or Amazon.

[24]E. Lawrence, "TLS and SSL in the real world," *IE Blog*, 21 Apr 2005.

HTTP, the result of this attack is indistinguishable without examining the page's source code. On login pages typically served over HTTPS, the user is relied on to realize the difference or to generally refuse to log into pages served over HTTP (which precludes using post-to-HTTPS sites).

Spoofing Browser Chrome: An important aspect of security indicators is that they are placed in the browser chrome, so the displayed cue is under the browser's control and not influenced by the content of the webpage being displayed (this has been called a 'trusted path' [110]). However as websites have been granted more power through client-side scripting and control over how a browser window is displayed, a variety of 'web spoofing' attacks [45] enable the website to interfere with how a user perceives the browser's security cues. For example, a well-positioned pop-up window without any chrome may overlap security cues on the underlying page [70] or simulate a browser window with complete browser chrome within the content of the page [15], [110], [58]. In one study, a site implementing the latter attack was classified as legitimate by 63% of users [58]. Today's desktop browsers typically force pop-up windows into new tabs, maintaining a constant chrome. In many mobile browsers, websites can position the address bar so that it is hidden, enabling spoofing of security indicators [44]. A solution suggested in the literature (but not commercially adopted) is a dynamically changing browser chrome [110], [35].

Users may also falsely attribute an HTTPS connection to a lock displayed somewhere other than the chrome, *e.g.,* on the page content [36] or in the site's favicon [73]. Of the Alexa top sites with login pages, 29 of 125 displayed a lock in the site content (including 70% of banks) or favicon [98].

Conceding a Warning: A MITM adversary may choose to substitute in a certificate with an untrusted chain and hope that users click-through or otherwise ignore the warning. This was exemplified in the previously mentioned attack attributed to the Syrian Telecom Ministry.

V. Security Enhancements to CA/B Model

Section IV reviewed the spectrum of issues with the CA/B trust model. Here we evaluate a collection of the most prominent among known proposals to enhance aspects of the trust model, deconstructing and evaluating their core ideas. A few of these ideas have been incorporated into one or two browsers platforms; others can be achieved with a browser extension. Instead of focusing on specific tools, we distill the main concepts behind each tool into a set of primitives that can be combined in different ways to address security issues within the CA/B model.

A summary is provided in Table I. The columns provide a framework for evaluation. The first set of columns, *Security Properties Offered*, show a set of properties not met by the current HTTPS and CA/B model but which selected primitives (as designated in the rows) provide. Primitives that offer a certain enhancement typically trade off aspects of security, privacy, deployability and usability. The next set of columns are used to evaluate the enhancement according to *Security & Privacy*, *Deployability*, and *Usability* (*cf.* [29]).

Combination Logic: It is interesting to consider how the primitives (rows) of Table I can be combined, to achieve broader sets of enhanced properties. In general, if the primitives of two rows are combined, the combined primitive inherits the strongest level of individual fulfillment from the *Security Properties Offered* columns (a logical-OR) but the weakest level from the *Evaluation of Impact on HTTPS* columns (a logical-AND).

A. Security Properties Offered by Primitives

Detecting Certificate Substitution (Table I–column A): Section IV-B provided several examples of how adversaries have illegitimately obtained browser-accepted certificates for subject domains (targets) they do not control. An adversary capable of modifying a TLS handshake intended for this target (through *e.g.,* wide-impact DNS hijacking or on-path interception near the server) could actively replace the target's certificate with his own—a substitution allowing read/write access to the encrypted content without triggering browser warnings or errors. Primitives that detect such a MITM attack involving a substituted certificate are listed under *Detects MITM* (first column, Table I). If a primitive requires risk or "blind" trust on first use (TOFU) to detect these attacks, we use ◦ to denote partial fulfillment.

Some primitives detect only specific subclasses of such MITM attacks. We say a MITM attack is *local* if the adversary is able to insert himself into connections to the server from only a subset of clients (through *e.g.,* poisoned local DNS cache or on-path interception near the client); if detectable, the primitive fulfils *Detects Local MITM*. An HTTPS connection is often used to transmit a client authentication credential (*e.g.,* a password or secure cookie) to the host. Some primitives focus on protecting against credential theft during an HTTPS MITM attack; these fulfill *Protects Client Credential*. Again, blind TOFU primitives partially fulfill (◦). Some primitives that use pinning (see below) make false-reject errors if a server updates its public key, switches issuing CAs, or uses multiple certificates for the same host. Primitives that resolve such false-reject errors fulfil *Updatable Pins*.

Detecting TLS Stripping (Table I–column B): Section IV-E outlined TLS stripping attacks where HTTPS POSTs [80] or GETs [73] are simply downgraded to HTTP by a man-in-the-middle adversary. Since many enhancements to HTTPS do not take into account security-relevant details of a connection until there is an HTTPS request from the client, TLS stripping bypasses them. Primitives that can detect stripping attacks fulfil *Detects TLS Stripping* and partially fulfil it if they rely on blind TOFU. Primitives that deter (through enforcement or a security indicator) POST

Table I
COMPARATIVE EVALUATION OF PRIMITIVES TO IMPROVE THE CA/BROWSER MODEL. PROPERTIES (COLUMNS) ARE POSITIONED AS BENEFICIAL WITH FULFILMENT DENOTED BY ● AND PARTIAL FULFILMENT BY ○ (FOR DETAILS, SEE SECTION V). RATINGS ARE FOR THE POTENTIAL OFFERED BY THE GENERIC PRIMITIVE AND MAY DIFFER FROM CONCRETE INSTANTIATIONS OF PROPOSALS.

Primitive	*Security Properties Offered*								*Evaluation of Impact on HTTPS*										
	A				B		C		*Security & Privacy*				*Deployability*				*Usability*		
	Detects MITM	Detects Local MITM	Protects Client Credential	Updatable Pins	Detects TLS Stripping	Affirms POST-to-HTTPS	Responsive Revocation	Intermediate CAs Visible	No New Trusted Entity	No New Traceability	Reduces Traceability	No New Auth'n Tokens	No Server-Side Changes	Deployable without DNSSEC	No Extra Communications	Internet Scalable	No False-Rejects	Status Signalled Completely	No New User Decisions
Key Pinning (Client History)	○	○	○						●	●		●	●	●	●	●			
Key Pinning (Server)	○	○	○						●	●				●	●	●	●		●
Key Pinning (Preloaded)	●	●	●	●					○	●		●	○	●	●		●	○	●
Key Pinning (DNS)	●	●	●	●					○	●	●		○		●	●	●	○	●
Multipath Probing		●		●								●	●	●		●		●	
Channel-bound Credentials			○						●	●		●		●	●	●	●	○	●
Credential-bound Channels			○						●	●		●		●	●	●	●	○	●
Key Agility/Manifest				●					●	●				●	●	●	●	●	●
HTTPS-only Pinning (Server)					○	○			●	●				●	●	●	●		●
HTTPS-only Pinning (Preloaded)				●	●	●			○	●		●	○	●	●		●	○	●
HTTPS-only Pinning (DNS)				●	●	●			○	●			○		●	●	●	○	●
Visual Cues for Secure POST						●			●	●		●	●	●		●		●	
Browser-stored CRL							●		○	●		●	●	●	●	●	●	●	●
Certificate Status Stapling							●		●	●	●			●	●	●	●	○	●
Short-lived Certificates							●		●	●	●	●		●	●	●	●	●	●
List of Active Certificates							●	●			●	●	●	●		●	●	●	●

requests from being submitted over HTTP fulfil *Affirms POST-to-HTTPS*.

PKI Improvements (Table I–column C): Sections IV-D and IV-C respectively described two general problems with the PKI infrastructure: the lack of reliable revocation and the hidden nature of intermediate CA certificates. We assume in evaluating the primitives that CRLs or OCSP responses are not available and examine their ability to otherwise detect a revoked certificate; primitives which do fulfill *Responsive Revocation*. Primitives fulfil *Intermediate CAs Visible* if every intermediate CA is visible to the user at any time.

B. Evaluation Criteria for Impact on HTTPS

Security & Privacy: Some primitives introduce new infrastructure elements, which include entities that contribute to the trust decision or are queried when establishing an HTTPS connection. A primitive not introducing any new trusted parties fulfills *No New Trusted Entity*, with partial fulfillment if the responsibilities of an already trusted party are expanded. If it does not introduce any new parties that will become aware of all (or a fraction of) sites a user visits over HTTPS, it fulfills *No New Traceability*. If it eliminates such a class of entities (such as OCSP responders) it fulfills *Reduces Traceability*.

In the current HTTPS model, servers authenticate themselves through certificates. Many primitives effectively introduce new server authentication tokens, like (see below) pins or signed OCSP responses, that are transmitted to the client. Generally procedures for issuing, updating, and revoking these new tokens must be established, as well as integrity protection. Primitives that do not introduce such tokens fulfil *No New Auth'n Tokens*.

Deployability: Primitives that do not change how web servers implement TLS and HTTPS have the greatest potential for deployment. Primitives that do not require any server involvement or code changes fulfill *No Server-side Changes*, while primitives that only require servers to participate in a way that does not involve changing any server code partially fulfills it. Some primitives rely on DNSSEC which has not been fully deployed; the others are *Deployable without DNSSEC*. If primitives do not introduce an extra communication round that blocks completion of the connection, they fulfill *No Extra Communications*. Finally, *Internet Scalable* systems could foreseeably support enrolment from all current HTTPS servers and potentially beyond.

Usability: Evaluating usability properties is difficult without conducting user studies. However, some objective usability properties can be determined from the design and knowledge of the operating environment. A primitive fulfills *No False-Rejects* if it does not reject legitimate server certificates. A primitive not fulfilling *No False-Rejects* requires the user (*e.g.,* through a warning dialogue) to distinguish false-rejects from an actual attack. Primitives fulfill *No New User Decisions* if they are automated and do not require users to respond correctly to new security cues or dialogues.

In the current model, an HTTPS connection succeeds, with a security indicator (closed lock), when a site certificate is browser-acceptable. For some primitives, connections may succeed for other reasons. For example, if a primitive is blind TOFU, an indication of trust could be attributed to it being the first use or because the behaviour matches what is expected. Depending on how these primitives are implemented, users either (i) cannot readily determine the reason for trust, (ii) are frequently warned, or (iii) a new security cue is introduced. The latter two would respectively impact *No False-Rejects* and *No New User Decision*. Instead of assuming how these primitives should be implemented, we identify them as not fulfilling *Status Signalled Completely*. We award a partial fulfillment if the basis of trust is not clear because server enrolment is optional, and thus a fallback trust mechanism may be also necessary.

C. Summary and Evaluation of Proposed Primitives

Here we describe the primitives summarized in Table I. Detailed justification of the ratings is provided in the Appendix A.

Key Pinning (Client History): Other than the browser trust anchors, validating an HTTPS server certificate is a stateless process, independent of any previous browser acceptable certificates seen for a particular site. A pinning primitive based on client history (also called inductive pinning) remembers the last browser-acceptable public key encountered for a particular site and warns the user if this information changes. This allows detection of certificate substitution attacks, even if the adversary has somehow obtained a browser-acceptable certificate—but only if the user has visited the site previously. Note that the term 'key pinning" (used here and below) is a slight misnomer, as the pin could specify anything from the entire certificate chain to a predicate applied over various certificate attributes to only the SubjectPublicKeyInfo field of the server certificate. Client-based key pinning is proposed in CertLock [94] (which pins the issuing CA country) and implemented in Firefox extension Certificate Patrol (which pins the entire chain and shows differences through a dialogue).

Key Pinning (Server): With client-side key pinning, a trade-off results from the level of granularity of certificate information being pinned. Servers are better positioned to themselves know which certificate attributes are likely to remain stable over time, and certificate rollovers are typically planned in advance. Server-asserted pinning allows the server to specify in an HTTPS header or TLS extension which certificate attributes to pin and for how long. One proposal, HPKP [11], suggests that servers specify a set of public keys (the SubjectPublicKeyInfo field of an X.509 certificate) of which one must be present in each interaction. A second proposal, TACK [13], is similar but organizes server keys under a new TACK signing key, to which the hostname is pinned by the client. The TACK key is used to sign server certificates and can allow revocation (see *Key Agility/Manifest* below).

Key Pinning (Preloaded): To avoid the blind TOFU approach of server-asserted key pinning, browser vendors could include a list of pins within the browser itself. Google Chrome currently pins a number of certificates for its own domains, as well as others by request.[25] Among other issues, this pinning allowed Chrome users to detect the earlier-mentioned MITM attack involving DigitNotar. The list could be populated by entities other than the browser vendor.

Key Pinning (DNS): Recall that some CAs offer DV certificates to sites that can demonstrate control over the DNS record for their hostname. If a CA does no more than look at a DNS record to validate ownership then couldn't clients do it instead, cutting out the CA? In practice, it is easier for an adversary to manipulate the client's view of DNS, which could rely on a cache or local resolver [95]. However with DNSSEC, records digitally signed by the name servers give clients the ability to validate records. The DNS-based Authentication of Named Entities (DANE) protocol [3] proposes that servers pin their public key in their DNSSEC record for clients to validate against.

We note one benefit of DNS-based pinning proposed [7] not captured in our framework. Even without DNSSEC, sites could pin certificate attributes such as acceptable CAs, for other CAs to reference if they are ever asked, perhaps illegitimately, to generate a certificate for the site. This offers protection against social engineering attacks that impersonate a domain owner to a CA.

Multipath Probing: Crowdsourcing is an approach with many technological applications. Applied to making trust decisions in HTTPS, crowdsourcing might take one of several forms. First, participants could contribute objective (*e.g.,* "I have seen this certificate before") or subjective (*e.g.,* "I do not trust the CA that issued this certificate") information. We cannot definitively evaluate subjective crowdsourcing as the properties it achieves depend on the quality of information provided by a client's peers. Tools that enable subjective trust assertions (whether crowd-sourced or from a delegated authority) include Omnibroker [9], Monkeysphere (web of trust PKI for HTTPS), YURLs (URLs with a built-in public key fingerprint obtained from a trusted peer), and S-Links (links to HTTPS sites where the links specify certificate information or extra validation steps).

Objective measurements generally fall along the dimen-

[25] A. Langley, "Public key pinning," *Imperial Violet* (blog), 04 May 2011.

sions of time (*e.g.,* "I see the same certificate as last time") and space ("I see the same certificate as my peer"). Time-based measurements are captured by inductive client pinning, however we note that the usefulness of blind TOFU can be extended by finding the peer with the earliest "first use." Multipath probing is a space-based measurement; the idea is to establish if the client receives a certificate that is consistent with the certificate received by independent observers (notaries) distributed across the internet. Multipath probing can detect local certificate substitution attacks, but not attacks where all traffic to the host is modified.

As a primitive, multipath probing was proposed in Perspectives [108], available as a Firefox extension. Convergence [74], also a Firefox extension, is a refinement that provides a more general architecture for crowdsourcing that could include subjective information as well as objective measures. Another refinement, DoubleCheck [18], probes from multiple servers in the existing Tor anonymity network (which cleanly eliminates the privacy threat notaries pose in terms of tracking). In addition to network devices, any collection of certificate data can additionally be consulted for an independent perspective (*e.g.,* the SSL Observatory [40] or ICSI Notary [19]). Certificate Transparency (CT) [6] is a proposal for creating a central audit log of HTTPS certificates, which is verifiably append-only and maintained by independent monitors.

Channel-bound Credentials: Some primitives forgo detecting MITM attacks in favour of protecting some of the information otherwise subject to theft by a MITM adversary. HTTPS is commonly used to provide secure transport of client authentication credentials, *e.g.,* passwords and cookies. Channel-bound credentials make such credentials functionally dependent on the specifics of the HTTPS connection—specifics an adversary cannot replicate even with a browser-acceptable certificate for the site. Rather than modifying user-chosen passwords, these primitives modify the authentication value in cookies. With channel-bound cookies [37], this value is cryptographically bound to a semi-persistent, site-specific public key certificate generated on-the-fly by the client, called an origin bound certificate (OBC). To login with such a cookie, the client first establishes a mutually authenticated TLS connection using its OBC and then transmits the OBC-dependent cookie. A MITM adversary cannot successfully use a stolen OBC-bound cookie unless it can establish an OBC-authenticated TLS session, which requires knowledge of the corresponding private key. OBCs were then revised (for details not pertinent to preventing MITM cookie theft) and renamed Channel ID. The channel-bound credential primitive has also been proposed for use in conjunction with a user device: *e.g.,* a token [79] or smartphone [82].

Credential-bound Channels: Credential-bound channels prevent credential theft from MITM adversaries by reversing the idea of channel-bound credentials—instead of having the server decide to accept a credential based on its binding to a client certificate, the client decides whether to accept a server certificate based on its binding to the client's credential. This primitive assumes a pre-shared password (and does not protect the password during the initial establishment). In one proposal called direct validation of certificates (DVCert)[26] [34], the server uses a PAKE-based protocol to demonstrate knowledge of the client's password while attesting to the value of its certificate.[27]

Key Agility/Manifest: When preventing MITM attacks without involving the server (*i.e.,* through inductive client pinning or multipath probing), it is difficult to distinguish attacks from legitimate reasons that a different certificate may be observed at different times (certificate update) or from different locations on the network (multiple certificates used by the same host). At the cost of server-side changes, many of the examples of the primitives evaluated thus far address these issues with (i) a "key manifest," *i.e.,* a specification of all keys that could be used by the domain; and/or key agility, *i.e.,* an update mechanism for new certificates that could be implemented as either (ii) signing the new certificate with the old certificate's key, or (iii) linking the certificate changes through use of a master secret. One or more of these can be seen in: server-side key pinning, TACK, DANE, and DVCert. The Sovereign Key proposal [12] is also similar to (iii), in that servers establish and broadcast a long-term signing key used to cross-certify all their certificates. Although Table I evaluates the primitive in isolation, in these examples, it is combined with another primitive, *e.g.,* key pinning or multipath probing, to detect MITM attacks while eliminating false-reject errors due to certificate updates and load-balancing.

HTTPS-only Pinning (All Types): The primitives considered above do not address (or attempt to) TLS stripping attacks. This is largely because the primitives are never invoked unless an HTTPS connection is requested. With TLS stripping, this stage is never reached for the client. The chief mechanism for preventing TLS stripping is to make certain domains only support TLS and communicate this to clients with a pin. As with key pins, HTTPS-only pins can be communicated by the server in request headers or TLS extensions, pre-established in the user's browser, or obtained from the DNS record of the site. Proposals for all three types exist: ForceHTTPS [57] and its refinement in HSTS [8] are server-initiated pins; Chrome 22 ships with over 100 HTTPS-only pins; and Service Security Requirement (SSR) [4], [81] records can specify (among other things) that a site is HTTPS-only in its DNS record (signed through DNSSEC). Some browser extensions, like HTTPS Everywhere, redirect the HTTPS version of sites according to a curated whilelist of domains.

[26]This is not to be confused with domain validation (DV) certificates.

[27]This is better than the server MACing its certificate with the shared password, which would admit an offline password dictionary attack.

Beyond TLS stripping, HTTPS-only pins can also ensure a cookie scoped to the domain of the pin is always sent over HTTPS, regardless of whether the website developer remembered to mark the cookies as secure [57], [68].

Visual Cues for Secure POST: A simple client-side primitive can address certain types of TLS stripping. Sites are frequently designed to cause login credentials to be POSTed to an HTTPS site from an HTTP site. A persistent security cue could be introduced to indicate if a form POSTs to HTTP or HTTPS (beyond the easily-disengaged warning upon an initial POST-to-HTTP). One proposal, the SSLight browser extension [92], adds a "traffic light" cue to login form fields that displays a green light if and only if the field is posted to the current domain over HTTPS (a yellow light indicates cross domain HTTPS posts and red indicates POST-to-HTTP). Note that the browser needs to retrieve the site certificates associated with all POSTs on the page, largely nullifying the main performance benefit from not serving the landing page over HTTPS to begin with. Further, the choice of displaying the cue in the login form field itself, which is part of the site content, risks a malicious modified site obscuring the real cue and spoofing its own (*cf.* [110]).

Browser-stored CRL: Four prominent long-standing revocation approaches are [78]: CRLs, online certificate status checking, short-lived certificates, and trusted directories. The CA/B model uses the first two through CRLs and OCSP respectively. Given the shortcomings of current revocation procedures (Section IV-D), attention to improving the responsiveness of revocation is being pursued along the lines of all four types. Browser-stored CRLs fall under the first (CRLs), modifying the architecture of CRL distribution from the current CA/B model. Instead of user clients fetching CRLs (and OCSP responses) directly from CRL distribution points, the browser vendor fetches these periodically and sends to the browser an updated master CRL for storage. All major browsers manually revoke high risk certificates through software updates, but Chrome has implemented a more general CRL that can be transparently updated.[28]

Certificate Status Stapling: In the same way that *Browser-stored CRL* is an architectural change to how CRLs are distributed in the current CA/B model, *Certificate Status Stapling* modifies the distribution of OCSP responses. Currently, a user client requests a status report from a CA-designated OCSP responder, but responders are often overwhelmed or do not usefully respond, in which case, the HTTPS connection typically completes without warning. Under *Certificate Status Stapling*, the certificate holders periodically obtain signed and timestamped status reports, and include these with their certificate during a handshake (*cf.* [89]). Within HTTPS, this is defined as a TLS extension [2] and commonly called OCSP-stapling. The RFC only permits a report on the server certificate, not the entire chain—a significant drawback. However Table I evaluates the general idea of stapling reports on the entire chain.

[28]A. Langley, "Revocation checking and Chrome's CRL," *Imperial Violet* (blog), 05 Feb 2012.

Short-lived Certificates: This primitive, representing the third type of revocation, replaces long-lasting certificates with short-lived ones that certificate holders frequently renew [89]. Revocation results from simply failing to update a certificate. In a recent proposal and implementation for short-lived HTTPS certificates [102], servers are issued certificates with a four-day lifespan (roughly the lifetime of common cached OCSP responses) either on-demand or in batches. It is proposed [102] to be used in conjunction with *Browser-stored CRL* and *Key Pinning (Server)*.

List of Active Certificates: The fourth revocation method is trusted directories, by which we mean a publicly searchable list of valid certificates. In HTTPS, this could be implemented as a whitelist of every server and CA TLS certificate (including intermediates) that is acceptable to the HTTPS clients that rely on this list. A certificate not on the list is not acceptable in any part of the certificate chain. Revocation is accomplished by removing the certificate from the list. This primitive makes visible all intermediate CAs, and also allows domain owners to monitor for illegitimately-issued certificates for their domains. No proposal for full-fledged *List of Active Certificates* has been made. It is similar to Certificate Transparency [6] (see above) but differs on the issue of revocation: the CT log is meant only as a reference for discovering suspicious certificates, not an authoritative whitelist for making trust decisions. Also CT currently logs site certificates only, which means intermediate CAs are visible only if they issue site certificates.

VI. Further Discussion and On-going Research

Further discussion here first mentions research areas orthogonal to our main focus on HTTPS and its certificate trust model. We then discuss primary areas of ongoing research.

1) Important Orthogonal Problems: The original objective [85] of HTTPS was to provide a confidential channel with message integrity and server authentication. However HTTPS does not bridge the cognitive gap (exposed by phishing, even with the presence of TLS certificates) between the user's cognitive notion of what organization they intend to connect with, and the domain name within the content of a certificate—and the latter is the only information in DV certificates that can be relied on. (It is interesting to ask: does identification of a syntactic domain name deliver the desired property of "server authentication"?) As for the extremes, many websites do not use TLS at all, while for the minority that use EV-SSL certificates, the validated organizational details are arguably insufficient—they are not referenced by users, nor typically endorsed by CAs familiar to users, even if users did understand how certificates were meant to work. For these reasons, TLS in its present form fails to close this cognitive gap. It remains an open research problem to find methods to address this issue.

HTTPS can protect the secrecy and integrity of cookies in transit; browser policy dictates the conditions for read/write access to cookies marked as secure. Likewise, browsers must handle mixed content carefully, including how/when to alert users—*e.g.,* different policy decisions are needed to handle the mixing of secured and unsecured content according to three delivery mechanisms (no TLS, TLS, and EV TLS) and different content types (*e.g.,* images, objects, scripts). Cookie security and mixed content remain challenging problems.

A compromised client-platform (*e.g.,* due to malware) can subvert HTTPS protections, including how protections are communicated to the user. Research on building verified kernels, trusted modules, and/or trusted paths into client platforms therefore complements HTTPS. Other orthogonal issues include the availability of the HTTPS infrastructure (*e.g.,* DDOS attacks, restrictive networks, captive portals) and improving performance (*e.g.,* False Start, Snap Start).

2) Protocol-level TLS–Analysis and Modification: The complexity of TLS has significantly enabled protocol attacks, even after 15 years. Analysing TLS security in sufficiently broad models remains an open research problem. Among other challenges, designing a protocol with provable security is easier than proving the security of a fixed protocol like TLS; many proof techniques require simplifying assumptions or the ability to make at least minor modifications. Security analysis of the basic functionality of TLS [106], [76], [83], [53], [27], [77], [49], [48], [59], [51] and its use of cryptographic primitives [66], [61] has provided results both positive (security proofs) and negative (attacks).

The discovery of flaws in non-essential or little-used components of TLS has produced a culture of work-arounds by disabling features (*e.g.,* renegotiation, CBC-mode ciphersuites, and compression as cited earlier) rather than protocol redesign. Such quick fixes are at the expense of long-term protocol evolution: *e.g.,* as of 2011, 99.6% of HTTPS sites did not yet support TLS 1.1 or 1.2 [88].

Some aspects of TLS are agile, *e.g.,* AES adoption was "quick" [69] due to a pre-existing ability to negotiate blockciphers. Other aspects, however, are not agile, *e.g.,* the two hashes used for pseudorandomness (MD5 and SHA1) in TLS 1.0 are non-negotiable, impeding SHA2/3 adoption; and likewise, the RSA padding format used for key transport impedes OEAP support [86], [26]. An open challenge is how to expedite protocol upgrades; TLS 1.2 adds agility but had only a 0.02% adoption rate in 2011 [88].

3) Trust Model Infrastructure: Critical research questions remain regarding the CA/B trust model—is its continued use unavoidable, does it still solve more problems than it creates, or has it become a liability? In the real world, trust is transitive in at most short chains whereas the CA/B model allows long chains. But even in a chain involving only one intermediate CA, the end-user, *de facto*, ends up 'trusting' the browser vendor who sets hundreds of trust anchors in the browser, the corresponding anchor CAs for endorsing thousands of intermediate CAs, and these intermediate CAs for certifying millions of websites. The end-user, as a relying party, in many cases would have no knowledge of or business relationship with CAs involved in these chains, and the CAs are apparently not accountable to end-users for errors. It is also worth remembering that in the original CA/B model, there were only a handful of CAs.

For better or worse, we have achieved *spontaneity*, one of the original goals [85] of TLS—an online world with great convenience. In fact, users can now go online for the first time and without any personal choices, 'trust' millions of sites. It would be unreasonable to expect this in the physical world; is it a realistic reflection of trust in the digital world? Progress is being made—*e.g.,* on increasing the transparency of the anchor selection process (*e.g.,* Mozilla's policies and public discussion for CA inclusion), indexing active CAs on the public internet [40], and providing users with configurable (and potentially delegatable [72]) trust anchors.

4) Human Element and the Security User Interface: Once CA trust anchors are somehow configured, browsers can automate many HTTPS security decisions, while providing to users status indicators through its interface. However, for a range of 'soft' errors—*e.g.,* expired certificates, mismatched domains, mixed content/scripting, untrusted CAs (including self-signed)—it remains without consensus whether browsers should fail open (with or without an indicator), fail closed, or provide a warning dialogue. Research reviewed in Section IV-E indicates low user confidence in navigating the current set of indicators and warnings. Testing defaults, UI changes, and the wording of warning dialogues requires further research, as well as protocol 'ceremony' [42], [64], [84] analysis that includes humans.

The challenges in designing usable security interfaces hint at a deeper problem with users' mental models of HTTPS. Experts can understand the relationship between encryption and authentication in TLS, and that a lock icon does not indicate that a website is safe in all senses. The inability of the community to provide interfaces and/or mechanisms that average users can understand remains problematic.

5) Raising the Bar (or Just Moving It): Many of the practical security issues with today's CA/B infrastructure result from a lack of defence in depth—*e.g.,* the compromise of a single CA defeats many current deployments of the model. Many of the enhancements in Table I aim to add depth by addressing MITM attacks, TLS stripping attacks, and revocation issues. Combining several primitives into the current infrastructure (*e.g.,* in an attempt to fully address MITM) offers the advantage of increasing protection, at the cost of a patchwork with increased complexity. A different approach is to seek alternatives that replace the functionality of CAs outright; in fact, some primitives in the table might be viewed as doing this.

A currently high profile approach that might be viewed as doing this is DNSSEC-based pinning (*e.g.,* DANE), which provides an infrastructure for ubiquitous HTTPS, largely replacing the need for CA-issued DV certificates. (Of course, many questions remain regarding performance, caching, and packet inspection if this were to be widely deployed or enabled by default.) It could be argued that the serious consideration currently being given to DANE signals the degree to which trust in CA-validated certificates has slipped, rather than the strength of DANE as an alternative.

DNS pinning by itself falls short of fully addressing the original HTTPS goal related to authentication, namely to support the transport of sensitive data to only an *intended* party. As mentioned above in relation to phishing, the intended target is not often identified by a domain name in the user's mind, but rather by the user's conception of a real-world entity. The extent to which high-assurance extended validation certificates can be leveraged to provide recognizable assurances to users remains an open question.

VII. Concluding Remarks

Among our objectives, we hope to raise awareness of the number and breadth of past and on-going security issues with HTTPS and its certificate trust model, allowing independent determination of their relative severities and root causes. The sophistication and difficulty of attacking the TLS protocol directly has apparently shifted attention, over time, to the security of the CA/B infrastructure and its reliance on human factors. Indeed, in the research literature, it is becoming more common for threat models to assume an adversary possesses a valid certificate for a targeted site.

From our comparative evaluation of proposals for enhancing the certificate infrastructure, several research trends are apparent. Among these: a variety of pinning techniques propose adding defence in depth (a) against attacks involving fraudulent certificates, and (b) against SSL stripping attacks (which are deceptively simple); many primitives rely on a trusted initialization before providing protection; and the continuing failure of HTTPS support as it exists today to provide responsive certificate revocation information.

Replacing the CA infrastructure is often viewed by individuals in the computer industry as a way forward. This may not, however, improve things unless the current underlying weaknesses do not reappear. Though certificate issuance was historically independent of DNS, tighter integration of the certificate trust model with domain names, and potentially more tightly with DANE, has been a notable evolution. A reasonable question to ask is: what tangible benefit to users does the best-possible commercially viable certificate validation offer beyond a binding between domain names and public keys? The answer may dictate the future of CAs.

Acknowledgements. We thank those who provided detailed comments on the paper, including the anonymous referees, Carlisle Adams, Joseph Bonneau, Cormac Herley, Adam Langley, Tim Moses, Trevor Perrin, and Sid Stamm. We acknowledge funding from the Natural Science and Engineering Research Council (NSERC) through a PDF (first author), Canada Research Chair in Authentication and Computer Security (second), and NSERC ISSNet (both).

References

[1] RFC 5746: Transport layer security (TLS) renegotiation indication extension, 2010. III-D3
[2] RFC 6066: Transport layer security (TLS) extensions: Extension definitions, 2011. V-C
[3] RFC 6698: The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA, 2012. V-C
[4] Internet-Draft: Storing service security requirements in the domain name system, 2006. V-C
[5] Internet-Draft: Maximum version TLS cipher suites, 2011. III-D2
[6] Internet-Draft: Certificate transparency, 2012. V-C, V-C
[7] Internet-Draft: DNS certification authority authorization (CAA) resource record, 2012. V-C
[8] Internet-Draft: HTTP strict transport security (HSTS), 2012. V-C
[9] Internet-Draft: OmniBroker protocol, 2012. V-C
[10] Internet-Draft: Preventing cross-protocol attacks on the TLS protocol, 2012. III-D4
[11] Internet-Draft: Public key pinning extension for HTTP, 2012. V-C
[12] Internet-Draft: Sovereign key cryptography for internet domains, 2012. V-C
[13] Internet-Draft: Trust assertions for certificate keys (TACK), 2012. V-C
[14] O. Aciiçmez, W. Schindler, and Ç. K. Koç. Improving Brumley and Boneh timing attack on unprotected SSL implementations. In *CCS*, 2005. III-B2
[15] A. Adelsbach, S. Gajek, and J. Schwenk. Visual spoofing of SSL protected web sites and effective countermeasures. In *ISPEC*, 2005. IV-E
[16] D. Ahmad. Two years of broken crypto. *IEEE Security and Privacy*, 6(5), 2008. III-B1
[17] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy*, 2013. III-C4
[18] M. Alicherry and A. D. Keromytis. Doublecheck: Multi-path verification against man-in-the-middle attacks. In *ISCC*, 2009. V-C, A5
[19] B. Amann, M. Vallentin, S. Hall, and R. Sommer. Revisiting SSL: A large-scale study of the internet's most trusted protocol. Technical report, ICSI, 2012. IV-C, IV-D, V-C
[20] C. Amrutkar, P. Traynor, and P. van Oorschot. Measuring SSL indicators on mobile browsers: Extended life or end of the road? In *ISC*, 2012. IV-E, IV-E
[21] G. Bard. The vulnerability of SSL to chosen-plaintext attack. Technical Report 2004/111, IACR ePrint, 2004. III-C2
[22] G. V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. In *SECRYPT*, 2006. III-C2
[23] E. Barker and A. Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. Special Publication 800-131A, NIST, 2011. III-A1
[24] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, 1997. III-C2
[25] L. Bello and M. Bertachhini. Predictable PRNG in the vulnerable Debian OpenSSL package: the what and the how. In *DEFCON 16*, 2008. III-B1
[26] S. M. Bellovin and E. Rescorla. Deploying a new hash algorithm. In *NDSS*, 2006. III-A2, VI-2
[27] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Cryptographically verified implementations for TLS. In *CCS*, 2008. VI-2
[28] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *CRYPTO*, 1998. III-C1
[29] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In *IEEE Symp. Security & Privacy*, 2012. V

[30] B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. In *ESORICS*, 2011. III-B2

[31] D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX Security*, 2003. III-B2

[32] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TSL channel. In *CRYPTO*, 2003. III-C4

[33] J. Clark and P. C. van Oorschot. SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE Symposium on Security and Privacy*, 2013. (document)

[34] I. Dacosta, M. Ahamad, and P. Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *ESORICS*, 2012. V-C

[35] R. Dhamija and J. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS*, 2005. IV-E

[36] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI*, 2006. IV-E, IV-E

[37] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *USENIX Security*, 2012. V-C, A6

[38] H. Dobbertin. Crytanalysis of MD5 compress. In *EUROCRYPT (Rump Session Talk)*, 1996. III-A2

[39] T. Duong and J. Rizzo. Here come the $\oplus$ ninjas. In *Ekoparty*, 2011. III-C2

[40] P. Eckersley and J. Burns. Is the SSLiverse a safe place? In *Chaos Communication Congress*, 2010. IV-B, IV-C, V-C, VI-3

[41] Y. Elley, A. Anderson, S. Hanna, S. Mullan, R. Perlman, and S. Proctor. Building certification paths: Forward vs. reverse. In *NDSS*, 2001. IV-C

[42] C. Ellison. Ceremony design and analysis. Technical Report 2007/399, IACR ePrint, 2007. VI-4

[43] S. Fahl, M. Harbach, T. Muders, L. Baumgartner, B. Freisleben, and M. Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *CCS*, 2012. IV-A

[44] A. P. Felt and D. Wagner. Phishing on mobile devices. In *USEC*, 2007. IV-E

[45] E. Felten, D. Balfanz, D. Dean, and D. S. Wallach. Web spoofing: An internet con game. In *NISSC*, 1997. IV-E

[46] B. Fox and B. LaMacchia. Certificate revocation: Mechanics and meaning. In *Financial Cryptography*, 1997. IV-D

[47] B. Friedman, D. Hurley, D. C. Howe, E. Felten, and H. Nissenbaum. Users' conceptions of web security: A comparative study (short talk). In *CHI*, 2002. IV-E

[48] S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, and J. Schwenk. Universally composable security analysis of TLS. In *ProvSec*, 2008. VI-2

[49] S. Gajek, M. Manulis, A.-R. Sadeghi, and J. Schwenk. Provably secure browser-based user-aware mutual authentication over TLS. In *ASIACCS*, 2008. VI-2

[50] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, V. Shmatikov, and D. Boneh. The most dangerous code in the world: validating SSL certificates in non-browser software. In *CCS*, 2012. IV-A

[51] F. Giesen, F. Kohlar, and D. Stebila. On the security of TLS renegotiation. Technical Report 2012/630, IACR ePrint, 2012. III-D3, VI-2

[52] I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr. Dobb's Journal*, 1996. III-B1

[53] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*, 2005. VI-2

[54] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security*, 2012. III-B1

[55] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements. In *IMC*, 2011. III-A2, IV-D

[56] C. Jackson and A. Barth. Beware of finer-grained origins. In *W2SP*, 2008. IV-A, IV-E

[57] C. Jackson and A. Barth. ForceHTTPS: Protecting high-security web sites from network attacks. In *WWW*, 2008. V-C

[58] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *USEC*, 2007. IV-E

[59] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO*, 2012. VI-2

[60] J. Jarmoc. SSL/TLS interception proxies and transitive trust. In *Black Hat Europe*, 2012. IV-B

[61] J. Jonsson and B. S. Kaliski Jr. On the security of RSA encryption in TLS. In *CRYPTO*, 2002. VI-2

[62] D. Kaminsky. Black Ops 2008: it's the end of the cache as we know it. In *Black Hat USA*, 2008. IV-A

[63] D. Kaminsky, M. L. Patterson, and L. Sassaman. PKI layer cake: New collision attacks against the global X.509 infrastructure. In *Financial Cryptography*, 2010. IV-A

[64] C. Karlof, J. Tygar, and D. Wagner. Conditioned-safe ceremonies and a user study of an appplication to web authentication. In *NDSS*, 2009. VI-4

[65] J. Kelsey. Compression and information leakage of plaintext. In *FSE*, 2002. III-C3

[66] H. Krawczyk. The order of encryption and authentication for protecting communications (or: how secure is SSL?). In *CRYPTO*, 2001. III-A2, VI-2

[67] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO*, 2010. III-A2

[68] A. Langley. Beyond the basics of HTTPS serving. *USENIX ;Login:*, Dec 2011. V-C

[69] H. K. Lee, T. Malkin, and E. Nahum. Cryptographic strength of SSL/TLS servers: Current and recent practices. In *IMC*, 2007. III-A1, VI-2

[70] S. Lefranc and D. Naccache. Cut-&-paste attacks with JAVA. In *ICISC*, 2002. IV-E

[71] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Public keys. In *CRYPTO*, 2012. III-B1

[72] M. Marlinspike. More tricks for defeating SSL in practice. In *DEFCON 17*, 2009. IV-A, IV-D, IV-E, VI-3

[73] M. Marlinspike. New tricks for defeating SSL in practice. In *Black Hat DC*, 2009. IV-E, V-A

[74] M. Marlinspike. SSL and the future of authenticity. In *Black Hat USA*, 2011. V-C, A5

[75] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In *CCS*, 2012. III-D4

[76] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *USENIX Security*, 1998. VI-2

[77] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, 2008. VI-2

[78] M. Myers. Revocation: Options and challenges. In *Financial Cryptography*, 1998. V-C

[79] R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication. *Computer Communications*, 29(12), 2006. V-C

[80] A. Ornaghi and M. Valleri. Man in the middle attacks: demos. In *Black Hat USA*, 2003. IV-E, V-A

[81] A. Ozment, S. E. Schecter, and R. Dhamija. Web sites should not need to rely on users to secure communications. In *W3C Workshop on Usability and Transparency of Web Authentication*, 2006. V-C

[82] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *Financial Cryptography*, 2006. V-C

[83] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM TISSEC*, 1999. VI-2

[84] K. Radke, C. Boyd, J. G. Nieto, and M. Brereton. Ceremony analysis: Strengths and weaknesses. In *IFIP SEC*, 2011. IV-B, VI-4

[85] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001. II, VI-1, VI-3

[86] E. Rescorla. Stone knives and bear skins: Why does the internet still run on pre-historic cryptography? In *INDOCRYPT (Invited talk)*, 2006. VI-2

[87] I. Ristic. Internet SSL survey 2010. In *Black Hat USA*, 2010. 3, IV-D

[88] I. Ristic and M. Small. A study of what really breaks SSL. In *Hack in the Box*, 2011. VI-2

[89] R. Rivest. Can we eliminate certificate revocation lists? In *Financial Cryptography*, 1998. V-C, V-C

[90] J. Rizzo and T. Duong. The crime attack. In *Ekoparty*, 2012. III-C3

[91] S. E. Schecter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's

new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *IEEE Symposium on Security and Privacy*, 2007. IV-E, IV-E

[92] D. Shin and R. Lopes. An empirical study of visual security cues to prevent the SSLstripping attack. In *ACSAC*, 2011. V-C

[93] J. Sobey, R. Biddle, P. van Oorschot, and A. S. Patrick. Exploring user reactions to new browser cues for extended validation certificates. In *ESORICS*, 2008. IV-E

[94] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. In *Financial Cryptography*, 2011. IV-B, V-C

[95] S. Son and V. Shmatikov. The hitchhiker's guide to DNS cache poisoning. In *SECURECOMM*, 2010. IV-A, V-C

[96] A. Sotirakopoulos, K. Hawkey, and K. Beznosov. On the challenges in usable security lab studies: Lessons learned from replicating a study on SSL warnings. In *SOUPS*, 2011. IV-E

[97] A. Sotirov and M. Zusman. Breaking the security myths of extended validation SSL certificates. In *Black Hat USA*, 2009. IV-A, IV-E

[98] D. Stebila. Reinforcing bad behaviour: the misuse of security indicators on popular websites. In *OZCHI*, 2010. IV-E, IV-E

[99] M. Stevens, A. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *EUROCRYPT*, 2007. III-A2

[100] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO*, 2009. III-A2

[101] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX Security*, 2009. IV-E

[102] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh. Toward short-lived certificates. In *W2SP*, 2012. IV-D, V-C

[103] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12:1–28, 1999. III-A2

[104] S. Vaudenay. Security flaws induced by CBC padding: applications to SSL, IPSEC, WTLS, ... In *EUROCRYPT*, 2002. III-C4

[105] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux. The inconvenient truth about web certificates. In *WEIS*, 2011. 3

[106] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *USENIX Workshop on Electronic Commerce*, 1996. III-D4, VI-2

[107] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter. The multi-principal OS construction of the Gazelle web browser. In *USENIX Security*, 2009. IV-E

[108] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Tech*, 2008. V-C

[109] T. Whalen and K. M. Inkpen. Gathering evidence: Use of visual security cues in web browsers. In *Graphics Interface*, 2005. IV-E

[110] Z. Ye, S. Smith, and D. Anthony. Trusted paths for browsers. *ACM TISSEC*, 8(2), 2005. IV-E, V-C

[111] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *IMC*, 2009. III-B1

[112] M. Zusman. Criminal charges are not pursued: Hacking PKI. In *DEFCON 17*, 2009. IV-A

[113] M. Zusman and A. Sotirov. Sub-prime PKI: Attacking extended validation SSL. Technical report, Black Hat Security Briefings, 2009. IV-A, IV-E

Appendix

A. Security Enhancements: Detailed Evaluation

In this section, we provide the detailed evaluation of the primitives summarized in Section V-C.

1) Key Pinning (Client History): As this primitive is blind TOFU, it only partially fulfils *Detects MITM*; and thus also partially fulfills *Detects Local MITM* and *Protects Client Credential*. It does not claim any other enhanced properties. Trust is not attributed to any new parties, fulfilling *No New Trusted Entity* and *No New Traceability*; it lacks *Reduces Traceability*. As the pins are not passed between entities as tokens, it fulfils *No New Auth'n Tokens*. There are *No Server-Side Changes*; it is available as an extension and thus *Deployable without DNSSEC*. It fulfils *No Extra Communications* and *Internet Scalable* as it only requires allocating and accessing a small amount of memory to store certificate information.

While pinning an entire certificate will detect the change of certificate used in an attack, it does so by detecting any change including legitimate certificate updates, a change of CAs, or the use of multiple certificates within the same site (*e.g.,* when a server farm load-balances traffic). Thus client pinning does not fulfil *No False-Rejects*. However pinning less granular information about the certificate (*e.g.,* as CertLock does) can reduce false-reject errors at the expense of false-accept errors, and thus in practice, *Detects MITM* can be traded-off with *No False-Rejects*. There is generally no user input but the frequency of false-reject errors would necessitate a warning dialogue, so it does not fulfill *No New User Decision*. It is up to the implementation to handle what user input, if any, is required when a user visits a site for the first time; *Status Signalled Completely* is unfulfilled.

2) Key Pinning (Server): Server-side pinning is blind TOFU and thus partially fulfils *Detects MITM*, and likewise *Detects Local MITM* and *Protects Client Credential*. Once a set of keys are pinned/tacked, servers are constrained in the types of on-demand updates that can be made to a certificate. The server-side pinning primitive itself does not fulfil *Updatable Pins* because it is possible to pin information that might change. This can be avoided by only pinning the issuing CA, reusing the same key in the new certificate, or waiting for the pin to expire. Further, when combined with *Key Agility/Manifest*, full agility is restored and updates become seemless (an approach used by TACK). Key Pinning (Server) does not claim any other enhanced properties.

As with client-side pinning, it fullfils *No New Trusted Entity* and *No New Traceability*, but not *Reduces Traceability*. Pins are effectively new authentication tokens, augmenting X.509 certificates, that themselves require an initial trust decision and benefit from expiration, update, and revocation mechanisms. Thus it does not fulfil *No New Auth'n Tokens*. It does not fulfil *No Server-Side Changes* as the HTTPS handshake is modified but it is *Deployable without DNSSEC*. Setting and comparing pins is relatively efficient; thus *No Extra Communications* and *Internet Scalable*.

Server-side pinning allows the same hostname to respond with multiple certificates, which avoids the load-balance warnings commonly experienced with client-side pinning (and *Multipath Probing* below) fulfilling *No False-Rejects*. Following the same rationale as *Key Pinning (Client)*, it fulfils *No New User Decision* but not *Status Signalled Completely*.

3) Key Pinning (Preloaded): We assume for evaluation the preloaded list is populated by the browser vendor. Preloaded pins fulfil *Detects MITM* and thus *Detects Local MITM* and *Protects Client Credential*. A browser's pre-established pins can be updated and pushed to users through a software update or other mechanism, fulfilling *Updatable Pins*. The primitive does not claim any other enhanced properties (but is often combined with HTTPS-only pinning; see below).

Browser-based preloads of pins only partially fulfils *No New Trusted Entity* as the browser, an entity already trusted to provide anchors, is further entrusted with attesting for specific certificates. Otherwise, it matches *Client Key Pinning* in fulfilling *No New Traceability* and *No New Auth'n Tokens* but not *Reduces Traceability*. It does not require any changes to how HTTPS is handled server-side, but servers do have to enrol by contacting the browser vendor; thus partial *No Server-Side Changes*. It is *Deployable without DNSSEC* and has *No Extra Communications*. It is not *Internet Scalable* if requests are made directly to the browser developers who must vet them.

It fulfils *No False-Rejects* and *No New User Decision* as per server-side pinning. Since including a pin in the browser is not mandatory, the implementation must decide if decisions based on pins will be differentiated from those that are not, hence *Status Signalled Completely* is only partially fulfilled.

4) Key Pinning (DNS): DNS-based key pinning offers a similar feature set as browser-based: *Detects MITM*, *Detects Local MITM*, *Protects Client Credential*, and *Updatable Pins*. It extends further trust to the DNS system: DNS registrars must ensure that updates to the record come from the domain owner, and DNSSEC brings its own PKI and trust anchors. Thus it only partially fulfils *No New Trusted Entity*. It fulfils *No New Traceability* and further, *Reduces Traceability*, by eliminating all OCSP responders for standard CAs.[29] DNS records become essentially an authentication token: *No New Auth'n Tokens* is unfulfilled.

Server enrolment is required, but not implementation changes, so it partially fulfils *No Server-Side Changes*. Since DNSSEC is not widely deployed, it does not fulfil *Deployable without DNSSEC*. It fulfils *No Extra Communications* since DNS records generally need to be retrieved anyway (the records can also be stapled into the handshake by the server). We note that the performance of DNSSEC for general DNS resolving is tangential to the task of relying on it specifically for establishing HTTPS connections. DNS pinning fulfils *Internet Scalable* as updating DNS records is already common practice. For the same reasons as browser-based pinning, DNS pinning fulfils *No False-Rejects* and *No New User Decision*, and partially fulfils *Status Signalled Completely*.

[29]Note there are no online revocation checks in DNSSEC infrastructure itself: revocation is handled through a REVOKE bit set and signed by other DNSSEC trust anchors (RFC 5011).

5) Multipath Probing: In our evaluation, we optimistically assume clients can establish a set of notaries with an honest majority. In this case, multipath probing fulfils *Detects Local MITM*. It also fulfils *Updatable Pins*, as (after a possible cache timeout) all notaries will see the same certificate. It does not offer any other enhanced properties. The notaries must be trusted: thus it does not fulfill *No New Trusted Entity*. While specific proposals, like Convergence [74] and DoubleCheck [18], contain anonymity enhancements to provide *No New Traceability*, by default, the primitive can track which sites a user visits over HTTPS: thus the primitive itself does not fulfil *No New Traceability* or *Reduces Traceability*. It fulfils *No New Auth'n Tokens* (notary responses are not tokens). Multipath probing fulfils *No Server-Side Changes*, *Deployable without DNSSEC* and *Internet Scalable*, however the communication round with all the notaries leaves *No Extra Communications* unfulfilled. Multipath probing generates false-reject errors when a host uses multiple certificates (*e.g.,* for load-balancing) and thus does not fulfil *No False-Rejects*. Users must decide which notaries to trust and must handle false-reject errors, so it does not fulfill *No New User Decision*. Multipath probing can be used with all sites, thus *Status Signalled Completely*.

6) Channel-bound Credentials: Channel-bound credentials fulfil *Protects Client Credential*, but only partially as the adversary could compromise any session where a server-assigned credential is not yet set (*e.g.,* initial login, after expiration or user deletion). It does not offer any other enhanced properties. There is *No New Trusted Entity*. As per our definition, there is *No New Traceability* as the primitive does not explicitly provide a new entity with information about what sites are being visited, however cookies in general do have privacy (and traceability in a different sense) implications (see [37] for a discussion of the privacy of origin-bound certificates). It further lacks *Reduces Traceability*. Channel-bound tokens require servers to extend TLS and modify cookie-handling, thus they fail *No Server-Side Changes*. They are *Deployable without DNSSEC*, *Internet Scalable*, and have *No Extra Communications* as the extra round in a mutually authenticated TLS handshake is comparatively negligible. It fulfils *No False-Rejects* and *No New User Decision*, but only partially *Status Signalled Completely* as channel-bound cookies might, initially, only be deployed by some servers.

7) Credential-bound Channels: Credential-bound channels partially fulfil *Protects Client Credential* since initial password establishment is unprotected. It does not offer other enhanced properties. Under the evaluation criteria, it matches channel-bound credentials in properties and generally by the same rational; while it requires a password to be entered, it still rates *No New User Decision* because the user would be entering a password anyway.

8) Key Agility/Manifest: This primitive fulfils *Updatable Pins*. There is *No New Trusted Entity* or *No New Traceability*, however it lacks *Reduces Traceability*. The certificate update chains and manifest are effectively tokens, so *No New Auth'n Tokens* is unfulfilled, as is *No Server-Side Changes*. However it is *Deployable without DNSSEC* with no *No Extra Communications* and *Internet Scalable*. It fulfils *No False-Rejects* (and can also fulfill this when combined with *Key Pinning (Client)* and *Multipath Probing*). This error reduction is only if it is widely deployed, and so it does not fulfil *Status Signalled Completely*. It delivers *No New User Decision*.

9) HTTPS-only Pinning (All Types): All three types of HTTPS-only pins fully fulfil *Detects TLS Stripping* and *Affirms POST-to-HTTPS* except server-initiated pinning, which only partially fulfills these due to blind TOFU. All except server-initiated pins fulfill *Updatable Pins* (as per the rational provided above with Key Pinning). Note that in fulfilling *Affirms POST-to-HTTPS*, we are assuming HTTPS-only pins have been activated on the destination. Since HTTPS-only pins are exactly like their key pin counterparts, the evaluation of *HTTPS-only Pinning (Server)* is identical to *Key Pinning (Server)*, and likewise between the *Preloaded* and *DNS* pin types (we thus do not repeat the analysis).

10) Visual Cues for Secure POST: Visual Cues for Secure POST fulfils *Affirms POST-to-HTTPS*. It introduces *No New Trusted Entity* and *No New Traceability* but lacks *Reduces Traceability*. It fulfils *No New Auth'n Tokens* and *No Server-Side Changes*. It is *Deployable without DNSSEC* and *Internet Scalable*. It does not fulfil *No Extra Communications* as certificates must be retrieved, whether or not an actual HTTPS link is followed by the user. An alternative implementation may optimistically assume any HTTPS link is secure, and then issue a warning if it turns out not to be, but such a cue cannot be interpreted to mean that it is secure to POST, only that it might be. As this primitive introduces a new cue that users need to be aware of, it does not fulfil *No False-Rejects* (users may make mistakes) nor *No New User Decision*. It does fulfil *Status Signalled Completely*.

11) Browser-stored CRL: This primitive fulfils *Responsive Revocation*. It partially fulfils *No New Trusted Entity* as the already-trusted browser is being further entrusted with revocation. It fulfils both *No New Traceability* and *Reduces Traceability*, the latter because OCSP responders can be eliminated. It fulfils *No New Auth'n Tokens*, *No Server-Side Changes*, is *Deployable without DNSSEC* and has *No Extra Communications*. Indeed, it improves performance by eliminating the need for an OCSP response (and considerably improves the non-default setting of failing closed under an inadequate response). It is *Internet Scalable*, fulfils *No False-Rejects*, is *No New User Decision*, and fulfils *Status Signalled Completely*.

12) Certificate Status Stapling: This primitive fulfils *Responsive Revocation*, *No New Trusted Entity*, *No New Traceability* and *Reduces Traceability*, the latter two because user clients no longer report visiting an HTTPS site to OCSP responders. It does not fulfil *No New Auth'n Tokens* as the status reports function like tokens, requiring integrity checks and validity periods. It does not fulfil *No Server-Side Changes*. It fulfils *Deployable without DNSSEC*, *No Extra Communications*, and *Internet Scalable*. It has *No False-Rejects*, is *No New User Decision*, but only partially fulfils *Status Signalled Completely* as browsers must decide how to distinguish servers without the primitive enabled.

13) Short-lived Certificates: Short-lived Certificates must trade-off shortening the certificates' validity period (which increases revocation responsiveness) and increasing server workload in obtaining new certificates. We therefore partially penalize *Short-lived Certificates* under *Responsive Revocation* for inability to revoke on-demand, still rating partial fulfillment (an improvement on *Vanilla HTTPS*).

The primitive fulfils *No New Trusted Entity*, *No New Traceability* and *Reduces Traceability*, the latter two because user clients no longer report visiting an HTTPS site to OCSP responders. It fulfils *No New Auth'n Tokens* as certificates remain the server authentication mechanism. It does not fulfil *No Server-Side Changes* as the HTTPS implementation does not change, but servers must obtain and install new certificates frequently. It fulfils *Deployable without DNSSEC*, *No Extra Communications*, and *Internet Scalable*. On usability, it has *No False-Rejects*, delivers *No New User Decision*, and *Status Signalled Completely*.

14) List of Active Certificates: This primitive fulfils *Responsive Revocation*. In general, it fulfills *Intermediate CAs Visible*, assuming that CA certificates are included as well as server certificates (which may not be the case in specific proposals). It does not fulfil *No New Trusted Entity* as the list maintainer must be trusted. Under the assumption that the list is too large to maintain locally, and requires searching, it does not fulfil *No New Traceability* or *Reduces Traceability*. It fulfils *No New Auth'n Tokens* and, in terms of deployability, *No Server-Side Changes* and *Deployable without DNSSEC*. Querying a list that's not stored by the user does not fulfil *No Extra Communications* but it is *Internet Scalable*. On usability, it has *No False-Rejects*, is *No New User Decision*, and *Status Signalled Completely*.