

Analyzing Proposals for Improving Authentication on the TLS/SSL-Protected Web

Christopher W. Brown¹ and Michael Jenkins²

¹ National Security Agency / U. S. Naval Academy, Annapolis, MD USA
wcbrown@usna.edu

² National Security Agency, Maryland, USA
mjjenki@tycho.ncsc.mil

Abstract. “Secure” web browsing with HTTPS uses TLS/SSL and X.509 certificates to provide authenticated, confidential communication between web clients and web servers. The authentication component of the system has a variety of weaknesses, which have led to a variety of proposals for improving the current environment. In this paper we survey, analyze, compare and contrast three prominent proposals. To do this, we attempt to systematically capture the properties one might require of such a system: authentication properties, forensics/privacy properties, usability properties, and pragmatic properties. Enumerating these properties is an important part of understanding these proposals and the nature of the authentication problem for the secure web. Finally, we offer a few conclusions and suggestions pertaining to these proposals, and possible future directions of research.

Keywords: web security, authentication, TLS, HTTPS, certificates.

1 Introduction

This article provides a summary and analysis of three proposals for improving the authentication component of the current environment for the TLS/SSL protected web (HTTPS) — specifically the client’s authentication of the server. As a part of this analysis, we identify the properties the overall system would, ideally, satisfy, and describe the various proposals in terms of the degree to which they do or do not provide/have these properties. The paper’s real contributions, hopefully, are a clearer picture of what the authentication problem for the TLS/SSL protected web really is, and a framework for evaluating new proposals both individually, and in combination with one another.

The current environment for secure web-browsing is based on TLS (see RFC5246 [1]). TLS provides a mechanism by which a web client (browser) and a webserver establishing a connection between one another make use of public key cryptography to agree on a shared secret key, which is then used to encrypt communication using symmetric encryption for the rest of that session. Typically the process begins with the client being furnished a domain name by the user, which is then translated to an IP address via DNS name resolution, after

which the client sends the ClientHello TLS message to the server presumed to be listening on port 443 at that IP Address. The client then receives messages from the server, one of which contains a public key. The client chooses a secret key (more accurately, a value that determines the secret key), encrypts it with the public key the client received, and sends the resulting ciphertext back to the server. At this point, both parties have the same secret key, and encrypted communication can commence.

This process guarantees that the client and the owner of the public key — i.e. the entity in control of the associated private key — are the only ones who know the secret key, assuming that the private key is kept private. It does not, however, guarantee anything about the identity of the owner of the private key. There is no assurance that the party the user of the client wanted to contact is the one with whom the client now has a secure connection. Routing to the IP address could have gone wrong. DNS resolution could have produced the wrong IP address. The domain name itself might be wrong — e.g., amazen.com instead of amazon.com. Thus, there is a critical authentication problem to be dealt with!

The current environment for secure web-browsing generally handles this authentication problem in the following (highly simplified!) way: 1) It is the user's responsibility to ensure that the domain name provided to the browser is in fact the correct domain name for the entity they are trying to contact. 2) The webserver sends its public key to the client in an X.509 certificate (see RFC5280 [2]) as part of a certificate chain in the TLS Server Certificate message, and it is the browser's responsibility to validate the certificate chain, and verify that it chains to one the certificates in the browser's list of trust anchors. 3) It is the responsibility of the CA that issued the trust anchor at the end of the certificate chain to ensure that the public key in the certificate really belongs to the entity that owns the given domain name.¹

It is assumed that the reader of this paper is familiar with this process, including details like certificate revocation lists, self-signed certificates, X.509 validation, etc. not covered in the very brief description above. It is also assumed that the reader is familiar with the manifold problems inherent in this system — for instance, that every CA trusted by the browser represents a single point of failure for the whole system, or realities of how users usually bypass the whole system [3,4]. The key point is that the authentication component of the current secure web-browsing environment has problems both in principle and in practice.

The many problems that exist in the current environment have prompted a wide variety of proposals for improvements. The proposals considered in this paper are limited to ones with a reasonable level of pragmatism (full scale replacement of the current environment with something new is extremely unlikely!), and a reasonable degree of visibility or momentum behind them. User interface improvements, while important, are also outside of the scope of this analysis. The proposals examined here are: DANE, Certificate Transparency and HTTP Pinning. We looked at the Sovereign Keys proposal closely, but it is not yet

¹ The responsibility for managing trust anchors falls to some combination of browser vendors, OS vendors, users and, perhaps, IT support departments.

mature enough to really analyze to the same extent as the above. Space restrictions forced us to remove our coverage of TACK [5], which is, however, similar in many respects to HTTP Pinning. Perspectives [6] and similar projects like Convergence, Google’s now defunct Certificate Catalog project, and the Berkeley ICSI project, are interesting as well, but space precludes covering them here.

This paper consists of three parts. The first describes the set of properties that might be desirable for the authentication component of the “secure web”. The second provides summaries of three proposals, along with some commentary and analysis — all of which is done in terms of the properties from the previous section. The last focuses on comparisons of different proposals and their potential to work in combination with one another.

2 Desirable Properties for the Authentication Component of the “Secure Web”

It is easier to understand and compare these various proposals if we first describe what it is we really want. In other words, What properties do we really desire of the authentication component of the “secure web”? What follows is a list of such properties, grouped into four categories. The fourth category is more accurately described as a list of properties we’d like to see in a proposal for improving on the current state of affairs, for example that the proposal is realistic. These properties were in large part deduced from reading a number of proposals and commentary on those proposals.

Authentication Properties²

continuity: that when a client connects to a host with name X, one can be sure that, in some meaningful sense, it is communicating with the same entity as it was communicating with the last time it connected to a host with name X. Note: it could make sense to think of continuity on two levels, the individual client level and the community level. Pinning proposals are about individual clients observing continuity. Notary proposals like Perspectives or the Berkeley ICSI, and gossiping proposals are about a community of clients observing continuity.

domain-name authentication: that the client is connected to the server authorized, intended or allowed to run under that name-and-port by the legitimate owner of that domain name.

higher-level authentication: that the client is connected to the server authorized, intended or allowed to run under that name-and-port by an entity described by some notion of identity beyond merely ownership of a given domain name. (e.g., Southwest Airlines, the U. S. Postal Service) There are actually different classes of certificates — EV (extended validation) [7], OV (Organization Validation), DV (Domain Validation) — that seek to provide some higher-level authentication. NOTE: Many companies have made their

² This article is only concerned with *server* authentication, so client-authentication properties are not addressed.

domain name their identity, e.g., amazon.com, so that domain-name identity and higher-level identity are one and the same.

attribute authentication: that the client is connected to the server authorized, intended or allowed to run under that name-and-port by an entity with certain attributes (e.g., FDIC insured bank, NASDAQ listed company).

Forensics and Privacy Properties

client privacy: that third parties cannot, without the willing participation of the client or the server, deduce what websites a given client has been connecting to (without special access to the client machine).

impostor discoverability: that the legitimate owner of a given domain name should be able to discover what servers are presenting themselves as belonging to that name.

server privacy: that third parties cannot, without the willing participation of the client or the server, deduce the existence of a given server without actually attempting a connection themselves.

local privacy: that someone with access to the client machine after a website has been visited cannot deduce facts about what sites have been visited by the client. Of course, if a proposal includes client-side data, a user should be able to “clear history” as they can with current mechanisms like cookies. However, this presumably degrades the improvements to authentication. So this property refers to privacy concerns assuming that this clearing of authentication history hasn’t happened.

Usability Properties

minimal false positives: that the client seldom refuses a connection or presents the user with an error message when the server it is connected to is indeed the right one. Ideally the client never decides to disbelieve that the entity with which it is communicating has the proper identity when, in fact, it does. This is the ideal. In reality, we can only try to minimize the number of false positives. As noted elsewhere, encountering many false positives conditions users to simply override-and-accept. Conversely, users that err on the side of caution are actually discouraged from connecting to legitimate servers. Note that this is described as a “false positive” because of the analogy to medicine — the test says there is a problem when, in fact, everything is OK.

maximal actionable information: that the client should have good, actionable information to present to the user, information that allows for good decisions. This includes information that allows for a reasonable plan of action in case the user decides not to override-and-accept.

minimal user trust: that the trust users must place in other entities is minimized. Note: this is a complex issue, since it involves not only how many entities need to be trusted, but also to what extent they are trusted, and how bad things could be if some number of them colluded.

user control: that the user is allowed to override decisions, determine who or what to trust, etc. This means the system must allow for it. Particular clients might not.

minimal server trust: that the trust server operators must place in other entities is minimized.

minimal server roadblocks: that setting up a TLS server is not overly burdensome. Already, lots of people and organizations have difficulty doing it right. Ideally improved authentication mechanisms shouldn't set up too many barriers, technical, logistical or financial, to organizations that want to set up websites that use them.

Implementation/Infrastructure/Pragmatics Properties

incremental: that there are benefits (progress towards other properties) for those that opt-in without requiring that everyone participates.

minimal-impact: that big changes to the internet architecture are not required. Client-only solutions, for instance are very low impact, as are server-only. When both have to change, when third-parties are involved, and so forth, the impact grows.

no-break: that things that currently work relying on relatively common practices (e.g., local network TLS connections, changing domain ownership, web-hosting services, user-level TLS servers on hosts, etc.) do not break under the new proposal.

scalable/maintainable/robust: that the system works at internet scale, and can function over a long period of time. For example, if keys are lost or no longer secure: can they be changed? The system should function reasonably in the face of outages and attacks.

resource-friendly: that adopting the proposal does not slow communication too much, or require too much CPU time or memory. Resource-constrained mobile devices must be able to participate.

realistic: that the proposal does not make unreasonable assumptions or demands on individuals or society. For example, expecting organizations with no suitable financial or other incentive to run big servers might not be realistic.

In each of the following three sections we briefly describe a well-known proposal for improving authentication on the secure web, and analyze that proposal in terms of the properties described above. Each property is listed, along with commentary on the proposal in terms of that property. A small “gauge” icon accompanies each property to give a quick indication of whether the proposal positively or negatively affects a certain property, or has no substantial impact. For the first three categories — Authentication, Forensics and Privacy, and Usability — the gauge value is to be understood as relative to the current secure web environment. For the last category — Implementation/Infrastructure/Pragmatics — gauge values are to be understood as relative to all the other proposals for improvements. The gauge values are simply manifestations of qualitative judgments, not true quantitative data. The hope is that when combined in tables as in the following sections, they are a concise means to provide insight into the strengths, weaknesses, and purpose of any given proposal. However, it is the accompanying commentary that provides the actual analysis.

3 DNS-Based Authentication of Named Entities (DANE)

DNS-Based Authentication of Named Entities (DANE) uses DNSSEC to make assertions that constrain valid certificate chains. These assertions can specify the end entity certificate or public key, the trust anchor certificate or public key, or an intermediate certificate or public key. By using DNSSEC to distribute these assertions, clients can guarantee that the assertions really belong to the domain name in question. Thus, DANE is a mechanism that provides very strong *domain name authentication*. DANE depends on DNSSEC, and DNSSEC adoption seems to be proceeding very slowly. DANE (see RFC6698 [8]) adds a new record type to DNS, the TLSA resource record, which allows the nameserver that is authoritative for a given domain name to make assertions tied to a pairing of the name with a port-and-protocol. These assertions are of the following form:

(usage, selector, matching type, certificate association data)
















where $usage \in \{0, 1, 2, 3\}$, $selector \in \{0, 1\}$, $matching\ type \in \{0, 1\}$, and the “certificate association data” can be a certificate, the Public KeyInfoField of a certificate, or a hash value.

The semantics of DANE assertions are essentially this: The certificate chain is constrained in usage 0 to contain a given certificate/public key, and in usage 1 to start with a given certificate/public key (which specifies the end-entity). For both, the client is still required to validate the certificate chain up to a client-trusted anchor. Usages 2 and 3 are similar, except that the client’s role in determining trust is eliminated. The certificate chain is constrained in usage 2 to contain a given certificate/public key as trust anchor, and that trust anchor must be trusted by the client for purposes of that validation, regardless of the client’s current trust store. The certificate chain is constrained in usage 3 to have a given end-entity certificate/public key, and if it does the client must accept the connection without doing certification path validation of the chain.

Because it is tied to DNSSEC, DANE’s pragmatic outlook is tied to DNSSEC adoption. It’s worth clarifying the sense in which DANE offers something more than DNSSEC alone would offer. A client relying on DNSSEC to resolve a given hostname to an IP address has a strong guarantee that the IP address it uses for that domain name is correct. However, it has no guarantees that the entity it connects to is a really a host properly associated with that IP address. After all, the attacker could be corrupting routing rather than DNS. With DANE, however, the client has stronger guarantees that the host it is connected to is properly associated with the given domain name.

Usage 2 & 3 assertions are potentially problematic. With usage 3, certification path validation does not occur, i.e. if the end-entity certificate presented to the client matches the certificate in the DANE assertion the connection is accepted. A usage 2 DANE response mandates a certain trust anchor for validation, and mandates that it be trusted — regardless of whether it is currently in the client’s trust store. Both deny the user (or system administrator) the opportunity to make trust decisions. The security issue here is that without usages 2 & 3 both DNSSEC/DANE and the CA system have to be defeated in a successful

attack, while with them an attacker that can successfully subvert DNSSEC can successfully pull off a man-in-the-middle attack. Defense in depth has been lost.

-  **continuity:** —
-  **domain-name auth.:** this is DANE's strength.
-  **higher-level auth.:** a usage 2&3 response bypasses certification path validation, so information in certificates is less trustworthy than in the current system.
-  **attribute auth.:** —
-  **client privacy:** generally, having to contact a third party server is a client privacy concern. However, clients would almost always be contacting a DNS server for name resolution anyway, so it's not really a concern here.
-  **impostor discoverability:** —
-  **server privacy:** normally a DNS server would store the IP address associated with a given name, and nothing more. DANE records include a port as well as IP, so the fact that a particular host is running a TLS server at a particular port number is then known to the DNS server.
-  **local privacy:** —
-  **minimal false positives:** DANE provides mechanisms (usages 3 & 4) by which a certificate that does not chain to a trust anchor would be accepted without any error or warning, which reduces false positives ... although it can also defeat authentication, as pointed out above.
-  **maximum actionable info:** if a site uses DANE and the client issues an error, the DANE assertion itself provides extra information about what public key / certificate should have been expected.
-  **minimal user trust:** the user doesn't need to trust the CAs as much, but they put more trust in DNS.
-  **user control:** with usages 2 & 3, there are situations in which a connection will be accepted regardless of the user's trust anchor settings.
-  **minimal server trust:** with DANE, the webserver operator puts even more trust in the DNS, however CAs don't need to be trusted as much.
-  **minimal server roadblocks:** to use DANE the webserver operator requires the cooperation of the DNS administrator. To also ensure that clients that do not support DANE aren't locked out, a certificate from a CA would also be required.
-  **incremental:** both web clients and sites (though not really webserver) need to change for DANE to work, any pair that supports DANE gain the security of the system, regardless of whether it's adopted elsewhere. However, it's not enough for just one of the two parties to elect to participate.

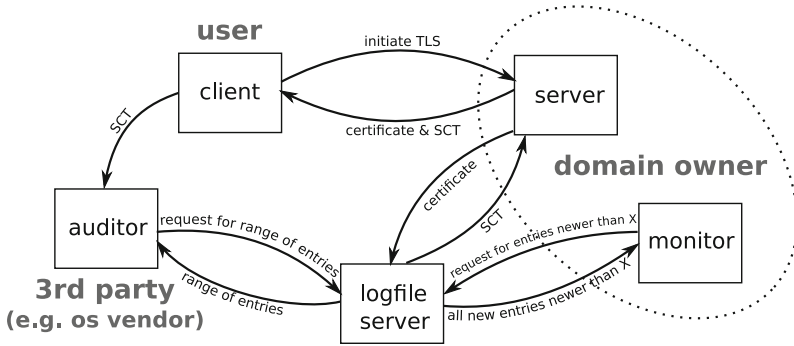







Fig. 1. Diagram illustrating CT. A full picture would show multiple logfile servers.

-  **minimal-impact:** it's not enough for client and server to change, the infrastructure has to change if DNSSEC is not already available.
-  **no-break:** —
-  **scal./maint./robust:** —
-  **resource-friendly:** DNSSEC has some overhead (see [9]) and more signed info will be sent when DANE is used than would be sent using DNSSEC solely for name resolution.
-  **realistic:** given that DNSSEC is in use, DANE is quite realistic. The question is whether it's realistic to expect significant DNSSEC adoption anytime soon.

4 Certificate Transparency

Certificate Transparency is described in RFC6962 [10]. Its primary purpose is to provide “impostor discoverability”. The basic idea is this: If there was a public logfile of all certificates issued, then domain name holders could view the public logfile to root out bogus certificates for their domain names and, as the proposal puts it, “invoke existing business mechanisms for dealing with misissued certificates”. If TLS clients all agree to reject any certificate not recorded in the public logfile, attackers would be forced to record their forged certificates in the logfile where, hopefully, server/domain owners would observe the bogus certs and do something about it. While this basic idea is straightforward, realizing it in a secure way is non-trivial. The Certificate Transparency proposal is somewhat complicated in terms of the number of entities involved: in addition to servers and clients, there are logfile servers, trusted auditors, and logfile monitors (see Figure 1). So the description that follows adds these various pieces in small steps.

Step 1: We first consider how clients determine whether a certificate record is in the logfile. Of course the client could contact the logfile server and ask.

Even if that could be done in an utterly secure and authenticated manner, there are still two issues: 1) contacting the third party has a performance and availability concern, and 2) letting the logfile server know every domain name you want to contact has privacy concerns. Therefore, the proposal calls for a different approach. Clients have preloaded/out-of-band-received public keys for the logfile server. The TLS server is supposed to send a “Signed Certificate Timestamp” (SCT) along with the certificate, which is essentially a hash of the certificate concatenated with a timestamp, signed by the logfile server. This gives the client something that it can verify quickly, without any third-party communication, so it addresses both concerns 1 and 2. IANA has issued a value for the TLS SCT extension. For technical reasons beyond the scope of this overview, the SCT is issued before the certificate is logged. However, the SCT contains an additional field with a value called the Maximum Merge Delay (MMD). Implicit in the SCT is a promise by the logfile server that the time between when the SCT was issued and when it is logged will not exceed that MMD value.

Step 2: If the logfile is misbehaving, or if it has been compromised, or its private key stolen or broken, clients could get forged SCTs. In other words, they could be accepting certificates that weren’t actually logged. To address this, the proposal calls for “trusted auditors” that clients are supposed to submit SCTs to, in order to keep tabs on the logfile server and make sure it really is reporting the submitted SCT as part of the log. The RFC mentions having the client do this (asynchronously, so as not to take the performance hit), but 1) that has all the same privacy concerns, and 2) the logfile server could systematically lie to that one host. So it makes more sense to introduce trusted auditors into the system. It’s unclear who or what auditors are notifying in case they detect a misbehaving logfile server, nor is it clear what the plan would be were a logfile server discovered to be misbehaving. Recovering from that situation could be quite challenging.

Step 3: Another way that a logfile server could misbehave is by modifying past entries in the log. For instance, maybe a bogus certificate gets a real SCT from the logfile server and is in the logfile (so the auditor doesn’t see any trouble) but then after the attack the logfile entry gets erased, and all this happens before the domain name owner gets a chance to check for any new entries in the logfile for his domain name. This would defeat the whole purpose of certificate transparency. Therefore, the logfile is *append-only* — once an entry is there, it’s there forever. This is done with Merkle trees, which provide a mechanism whereby anyone observing the logfile server could detect modifications or erasures of past entries. Of course, some entity has to bother to make these checks, so the proposal calls for “logfile monitors”, which would periodically query the logfile server and check that it was really operating in append-only mode. These might do double duty by also checking for new logfile entries for domains the host is interested in.

Step 4: Finally, the proposal envisages not one, but multiple logfile servers. To protect against Denial of Service attacks, in which the attacker floods a

logfile server with bogus certificates to be logged, the proposal suggests that each logfile server would publish a list of root CAs, and it would only log entries that validate via a chain up to one of the CAs in the list. The proposal also calls for “gossiping” to root out misbehaving logfile servers. However, no details on the gossiping protocol are given.











Certificate Transparency provides impostor discoverability. This is a benefit to server/domain-name owners, but only secondarily a benefit to users. It provides no benefit for the initial targets of attacks, but it does offer a potential benefit to the larger user community, in as much as a vigilant server/domain-name owner may notice the attacker and take steps to shut him down. The proposal takes great pains to ensure that entities that care to do so can monitor the activities of logfile servers in order to ensure that they are being honest, so that logfile operators don’t need to be trusted. There are, however, some issues to consider.

The purpose of the logfile monitors is to ensure that the logfile servers behave properly. However, once again, it is unclear how to deal with a logfile server that has misbehaved. It could be blacklisted somehow, but it’s not clear what to make of the SCTs it had previously issued. Webservers would be sending them out, potentially unaware that it was no longer trusted. Perhaps a bigger question is what to do with logfile servers that are not purposefully acting badly, but fail to meet an obligation — for example a logfile server that does not get the SCT into the log within the window specified by the MMD because of an attack, or a simple programming or administrative error. Simply blacklisting such a server seems highly undesirable. An alternative would be to rollback the log to the point of the error, but that’s a problem because all the legitimate SCTs that had been issued in between issuing this SCT and noticing that the merge deadline had been missed would then be invalidated. Some mechanism is required to deal with this gracefully.











The proposal doesn’t address how logfile public keys are distributed and updated. It seems that we end up in a similar situation as with CAs, namely that some arbitrary list of trusted logfile servers is preloaded into the browser. There is then the potential for even more certificate-related error messages, since a client could receive a certificate that is actually OK, but receive an SCT along with it that refers to a logfile server that is not trusted by the client (or for which the public key stored in the client is too old or too new).

It is not clear why users/clients would opt to submit SCTs to auditors. Collectively, there is the benefit that attackers could be discovered and eventually dealt with. But for the individual user there is little short-term benefit, and there is definitely a risk to privacy. If the intended model is that browser vendors would run their own trusted auditors, the privacy issue is mitigated, since their users are essentially trusting them completely anyway ... at least for Google, Microsoft and Apple. Less so for open source projects like Firefox, where users may put their faith in the “many eyes” that are supposedly on the source code. How a “trusted auditor” run by the Mozilla foundation is set up would not be subject to all those eyes, making it easier for a single individual or small group to

misuse it than to introduce errors into the Firefox codebase. Below is a summary analysis of CT.³

- + **continuity**: no real first-order effect.
- + **domain-name authentication**: no real first-order effect.
- + **higher-level authentication**: —
- + **attribute authentication**: —
- + **client privacy** : the auditor sees every secure site the client connects to.
- + **impostor discoverability** : this is the whole point of CT!
- + **server privacy** : a legitimate server has to announce its presence by submitting to a logfile server.
- + **local privacy**: no additional data is stored locally.
- + **minimal false positives** : with CT users could be faced with errors when valid certificates aren't logged, or when SCTs are sent to clients that don't have that logfile server's public key.
- + **maximal actionable information** : when a cert's trust anchor is untrusted by the client, but the cert is logged, the user at least knows that the cert has been available for scrutiny and for how long. Otherwise, the user will know that it is unlogged (which is more suspicious).

³ Draft revisions of the RFC address some of the issues raised here. This evaluation notes server privacy as an issue — a legitimate server needs to announce its presence by submitting its certificate to a public logfile server. To address this, draft revision 3 of the RFC includes a mechanism for redacting portions of the domain name in the certificate information submitted to a logfile server. For example, if the domain name in the certificate was `super.secret.example.com`, the information submitted to the logfile server might be `(PRIVATE).example.com`. Another mechanism added in the draft that addresses this problem is logging a name-constrained intermediate authority, along with a field that explicitly allows the SCT for the intermediate authority to stand in lieu of an SCT for the end-entity certificate. Thus, the situation above might be handled by having `super.secret.example.com` send an SCT for an intermediate CA constrained to `.example.com`. Concerns raised here regarding the Minimal Impact and Minimal Server Roadblocks Properties are addressed in draft revision 3 by providing a mechanism for including a server's SCT in its certificate. This way, the server/server owner does not necessarily need to change or do anything different in order for CT to be used. Instead, CA's could make sure SCTs are bundled in the certificates they issue, and servers simply send those certificates as they always do. Of course, this creates a chicken-and-egg problem: the CA needs the SCT to create the certificate, but the logfile server needs the certificate to create the SCT. To deal with this, draft revision 3 allows CA's to submit "pre-certificates" to the logfile server, which contain enough information for the logfile server to create an SCT. The SCT gets sent back to the CA, which then can complete the certificate. Because these mechanisms are only described in draft revisions under very active development, we are not including them in our analysis.

- + **minimal user trust** : on one hand, CT means users don't need to place so much trust in CAs (that's the "transparency"), but since a logfile server could essentially blackball a site by refusing to issue a SCT for it, users have to trust them to behave honorably. If there is one (or few) logfile servers for your client, that could become a problem.
- + **user control**: —
- + **minimal server trust** : server/domain owners don't need to place as much trust in CAs.
- + **minimal server roadblocks** : server/domain owners have to submit their certs to a logfile server, and have to find one that supports their trust anchor.
- + **incremental** : adoption is a major issue. If clients do participate, all sorts of legitimate sites will suddenly stop working, and users will get swamped in false positives.
- + **minimal-impact** : both clients and webservers need to change in order for CT to work, and a lot of additional infrastructure and new kinds of servers needs to be created.
- + **no-break** : how will this work in local network only situations? Will organizations be forced to run logfile servers inside their local networks? How about enterprise trust anchors? None of the usual logfile servers will support them, of course, so would such an enterprise need to run its own logfile server?
- + **scal./maint./robust** : lots of questions: what happens when logfiles make errors or are found to be acting improperly? How can logfile server keys change? How are logfile server keys distributed to clients? There are some significant maintenance problems!
- + **resource-friendly**: not a lot of extra burden on client or server; although clients have to report to auditors, they do it asynchronously.
- + **realistic** it is unclear what would motivate operators to stand up logfile servers, monitors or auditors.

5 An HTTP Extension for Public Key Pinning (HPKP)

At its most basic, "pinning" just means hard-coding or caching cert/public key (or the hash of the cert/public key) in a client, and requiring the cert/public key received at connection time to match what is currently "pinned". More flexibly, the client might pin a set of certs/public keys, or pin the cert/public key of an intermediate element of the certificate chain, both of which allow the end entity cert/public key to change in a controlled manner. Essentially, pinning is a commitment that the user won't allow certs/public keys to change. What's interesting is looking at the question of who controls whether, when and what pinning takes place. Pinning could be directed by 1) the user, 2) the client

(e.g., hard-coded pins, or pinning that would be updated by the client “calling home” or calling an external service, or a policy of caching certs/public keys after an initial unpinned connection), or 3) the server (directing pinning for itself or for subdomains). An example of (1) is when a user preloads or chooses to accept an ssh public key. An example of (2) is when applications (like Chrome) have preloaded pins or call back home to get new pinning directives. If a website were to send pinning directives to the client, that would be an example of (3). Pinning, obviously, is a mechanism for providing authentication *continuity*.

The IETF draft document draft-ietf-websec-key-pinning (at the time of this writing in revision 12) [11] proposes an HTTP extension (HPKP) that allows the server to direct the pinning performed by the client — i.e. it is an example of the category (3) type of pinning described above. In this proposal, the server sends clients HTTP directives (the proposed extension) providing (hash-algorithm,hash-of-public-key) pairs that are to be pinned. The client saves this pin information indexed by the domain name it used in creating the connection. On subsequent connections to the same name, the client then checks whether any hash value in the set of pins is matched by a hash of any of the public keys in the certifying chain. If so, the client continues as normal. If not, there is an error. A hash-algorithm + hash-of-public-key pair must be accompanied by a “max-age” value and may be accompanied by a “report-uri” value. The max-age value instructs the client to keep the pin for a certain time. The report-uri gives a URI that is to be used by clients to report pinning errors for that domain name. An additional assertion may be sent that directs the client to apply the pin not only to the server’s domain name, but to all of its subdomains as well.








The obvious benefit of HPKP is the *continuity authentication* it provides. When a user connects to a server often enough (meaning that the time between visits is less than the max age) with the same client, man-in-the-middle attacks should be detected. Because the server directs the pinning, and because sets of pins are allowed and intermediate public keys can be pinned, servers can pull off planned transitions to new public keys gracefully. As will be described in more detail below, the proposal is very good in terms of the Usability Properties. Among the Implementation/Infrastructure/Pragmatics properties, the only real concern is the extra resources required by a participating client — namely that a potentially large number of pins will have to be stored, which could be problematic for resource-constrained clients. There is an especial concern that a malicious site could flood the pin store and use up all the available space. The specification could perhaps be modified to bound the storage given a (non-top-level) domain, or reclaim space from the non-top-level domain with the largest storage footprint. Maintainability questions surrounding unplanned key transitions are answered by requiring servers to pin a “backup key”, which is a key to be used in case the current public key is compromised and needs to be revoked and its use discontinued.














Next we consider Forensics & Privacy Properties. What should be another obvious benefit of HPKP is *impostor discoverability*. This is, after all, the point of the reporting mechanism provided by the report-uri directive. But to what

degree will HPKP really provide this property? In the case of a man-in-the-middle attack (which is what an “impostor” really has to do), the client will be provided with a domain name X , and the attacker will somehow arrange things so that the client will think it is communicating with the host properly referred to by that name when, in fact, it is communicating with some other host — for example by disrupting routing. If the client receives a certificate chain and it doesn’t match what is pinned for the name X , the client is supposed to send information about the pinning error to the URI in the report-uri directive. However, it seems quite likely that this message will never arrive at its destination given that the attacker is already subverting network traffic to carry out the man-in-the-middle attack. So the only case in which this would actually have its desired effect is when the attacker was unwary enough to allow the reporting message through. We note that this could be remedied by having clients send these reports at exponentially decaying intervals — perhaps until a signed acknowledgment is received. As long as the attack is not permanent, the report should eventually get through. To avoid flooding-style attacks, a carefully analyzed approach that looks at domain relationships and drops multiple error reports from the same (non top level) parent domain should be investigated.

There are a variety of ways clients may end up with pins that don’t match the public key presented by a legitimate server. A domain name may change hands without the willing cooperation of the party losing the domain. Both primary and backup keys could be lost. An attacker manages to pull off a successful man-in-the-middle attack for a period of time on a site that doesn’t use pinning, and puts a “poison pin” in the browser of all clients that connect during that time, with a very big max-age. In all these cases, there’s actually a hole in the DNS namespace — a domain name that, for a large number of clients, is unusable for HTTPS connections. This is a potentially serious problem.

Finally, HPKP breaks with the general design principle of separating concerns, and the specific cryptographic principle that different security properties should be safeguarded by different keys (see, for example, Section 5.2 of [12]). TACK, which space precludes us from covering here, is a similar proposal, but it uses a separate key (the “Tack Signing Key”) to provide continuity authentication, and the certificate chain keys provide (as they are supposed to) domain name and higher-level authentication.

-  **continuity:** this is the point of HPKP, although the fact that pins expire limit this property.
-  **domain name auth:** —
-  **higher-level auth:** —
-  **attribute auth:** —
-  **client privacy:** attacks referenced above.
-  **impostor discoverability :** the report-uri directive provides this but, for reasons described above, it’s unclear how effectively.
-  **server privacy:** —

- + **local privacy:** there is forensics information in the pins themselves, and simply clearing the pinstore as you would the browser's cache is not an attractive option because the user would lose security.
- + **minimal false positives:** sites the user visits often shouldn't generate false positives, but those visited infrequently might; doesn't address first use.
- + **max actionable information:** for some errors, pinned information shows what public key user should expect to see, this can be actionable.
- + **minimal user trust:** the user trusts server X's pinning directives, but these only pertain to server X itself, so that's a pretty low level of trust. Pinning reduces the trust that must be placed in CAs.
- + **user control:** —
- + **minimal server trust:** a server making use of HPKP needs very little trust in CAs after a given client's first connection.
- + **minimal server roadblocks:** the server doesn't need to rely on, or coordinate with, outside entities to use HPKP in a limited way, but using it flexibly so that new keys can be introduced in a reasonable way may require getting a signing cert, which is a much bigger deal.
- + **incremental:** with a conforming client, any participating site is more secure (for the user), however, both client and server must participate.
- + **minimal-impact:** clients and web-servers need small modifications.
- + **no-break:** —
- + **scal./maint./robust:** some small concern about how domain names change hands.
- + **resource-friendly:** all the pin information needs to be stored, which could be problematic for memory-constrained clients. There are also concerns about attacks on memory resources via HPKP.
- + **realistic:** HPKP only requires buy-in from browser vendors to get started. Given that this is a Google draft, that buy-in might be there.

6 Comparisons and Conclusions

We have surveyed three proposals for for improving the current condition of authentication for the secure web, each of which are fairly well-known.

- DANE offers the prospect of providing strong guarantees of domain name authentication. However, with “usage values” 2 and 3 it eliminates the defense in depth that the system of certificate authorities was supposed to bring. Moreover, DANE is built on top of DNSSEC, and DNSSEC adoption has not progressed very quickly.
- Certificate Transparency offers a mechanism by which domain/server owners can detect attackers that are impersonating their sites. However, it has a

number of pragmatic problems, as detailed above, and may increase the number of “false positive” warnings experienced by users.

- The HTTP Extension for Server-directed Pinning (HPKP) is designed to provide continuity authentication. HPKP suffer from the “poison pin” problem, namely that once the wrong pin gets in a client’s pin store, the client will present the user with a “false positive” error message.

DANE (ignoring usage 2 and 3), Certificate Transparency and HPKP are pretty much orthogonal to one another, meaning that they could be used in combination without interfering with one another or overlapping in what they provide. In fact, used in conjunction we would have stronger domain name authentication (from DANE), continuity authentication (from HPKP) and improved impostor discoverability (from Certificate Transparency).

We finish up by looking beyond the proposals analyzed in this paper, and asking whether the analysis suggests new ideas to investigate or has any other interesting implications. The first thing we would like to point out is that instead of viewing proposals like these as trying to “fix” authentication for the TLS-protected web, we should evaluate a proposal by clearly understanding the authentication or forensics property it is trying to provide, and analyzing the extent to which it provides that property along with the positive and negative impacts on the other properties that would result from adopting the proposal.

A second point is that once we stop looking for a single, monolithic, universal fix to authentication for the TLS-protected web, the importance of “orthogonality” of proposals becomes quite clear, by which we mean that the adoption of a proposal wouldn’t interfere with existing mechanisms or other proposed improvements. When proposals are orthogonal they can be composed, and that strengthens authentication. DANE’s usages 2 and 3 are unfortunate precisely because they ruin orthogonality. Without usage 2 or 3, DANE and the current certificate infrastructure compose nicely.

A third point is that when we view these various mechanisms as providing evidence for one or more of the four authentication properties, we see each connection attempt as making a case for accepting the proposed identity of the server on the other end. It might be reasonable to pin the “shape” of that evidence — i.e. what kind of evidence was presented. So, for example, suppose a user has been using client-directed pinning, and DANE (without usage 2 or 3) is used in conjunction with the usual Certificate validation process. The user tries to go to `https://example.com` and there is an error — the public key in the Certificate presented by the server does not match what the browser has pinned for the name `example.com`. However, the DANE record is validated, and the certificate chains to a trust anchor. The decision about whether to trust this server despite the pin mismatch is unclear. Now suppose that on prior connections, `example.com` had sent the client an EV certificate, and suppose the client had pinned that fact. It would not be at all unreasonable to base the trust decision on whether or not the certificate presented by the server is an EV certificate. Note that pinning the “shape” of the authentication evidence provided by a site has the really nice property that it actually provides increased security to sites

that choose to employ stronger authentication evidence. In the example above, the client would have “pinned” the facts that example.com employs DANE and uses an EV certificate. Thus, an attack will generate a warning to the user unless the attacker both subverts DNSSEC and gets a fraudulent EV certificate for example.com. Taking control of DNS records might be enough to get an ordinary certificate for example.com, but it shouldn’t be enough to get an EV certificate. This kind of pinning could also make gradual adoption of some of these proposals easier. For example, if the client pinned the fact that a given site used Certificate Transparency (i.e. sent an SCT) in prior connections, then the client could be configured to *require* CT for that site from that point on, but not for other sites. This would eliminate the problem of clients being flooded with false positives. One of the interesting things about Perspectives [6] is that it explicitly presents itself as a mechanism for providing evidence about identity, not as a procedure that proscribes trust decisions. That’s a powerful and flexible idea.

The fourth and final point is a suggestion that we feel falls out of this analysis. We start with the observation that if clients were to do client-directed pinning of end-entity certificates and servers would do OCSP stapling, then most of the time there would be strong authentication of the server on the basis of those two pieces of evidence alone, and the connection could proceed⁴. The process would be quick and involve very little overhead provided that the certificates match, and the point is that they usually would. So the question really is how to deal with the infrequent situation in which the above is unable to confirm authentication. This can happen when a client connects with a given name for the first time, when a different end entity certificate is sent by the server, or when an end-entity certificate is revoked. Making the right decision in these cases, and doing it to the greatest extent possible without user intervention, is crucial. However, since these situations are infrequent (as well as important), when they do arise it would be acceptable to have the client take substantially longer to make a decision, or to gather information to present to the user in case it is necessary. We suggest research into mechanisms or the development of standards that allow the client to collect a lot of relevant data in order to make a strong case for or against trusting the server. As a very simple example, suppose there was a standard way for a client that was not able to authenticate the server using the pinning & stapling mechanism above, to fetch additional certificates for the server. A client could implement a policy requiring that, in this event, it is able to fetch an additional certificate that contains the same TLS key, the same domain name, and chains to a different trust anchor (without cross signing).

⁴ Client-directed pinning would ensure that the certificate hadn’t changed since the client’s last connection to the site, and OCSP stapling would ensure that the certificate had not been revoked — at least as of some reasonably recent point in the past. Online Certificate Status Protocol (OCSP) stapling is a piece of the modern certificate infrastructure. It allows a server to send clients a message, signed by the relevant certificate authority, that asserts that as of a certain point in time, the server’s certificate has not been revoked. This is a nice alternative to contacting OCSP servers to check for revocation, or either pushing or pulling blacklists.

This increases the difficulty of a man-in-the-middle attack significantly, since the attacker would have to obtain fraudulent certificates from two different trust anchors. This is orthogonal to other proposals, and it strengthens all of them. For example, HPKP has potential problems with unplanned key transitions. With a mechanism like this, an organization that is forced to deal with an unplanned key transition could have strong evidence (we've suggested multiple certificates as a possible form) that a client could fetch on that single connection for which HPKP broke. The client could be convinced with overwhelming evidence and accept the TLS connection — without user intervention. The delay caused by fetching and analyzing this extra evidence would only be incurred once, then HPKP would suffice for subsequent connections.

References

1. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.1. RFC 5246, RFC Editor (April 2006)
2. Housley, R., Santesson, S.: Update to directorystring processing in the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280, RFC Editor (August 2006)
3. Herley, C.: So long, and no thanks for the externalities: the rational rejection of security advice by users. In: Proceedings of the 2009 Workshop on New Security Paradigms Workshop, NSPW 2009, pp. 133–144. ACM, New York (2009)
4. Sunshine, J., Egelman, S., Almuhiemedi, H., Atri, N., Cranor, L.F.: Crying wolf: an empirical study of ssl warning effectiveness. In: Proceedings of the 18th Conference on USENIX Security Symposium, SSYM 2009, pp. 399–416. USENIX Association, Berkeley (2009)
5. Marlinspike, M., Perrin, T.: Trust assertions for certificate keys. Internet-Draft draft-perrin-tls-tack-02, IETF Secretariat (January 2013)
6. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: improving ssh-style host authentication with multi-path probing. In: USENIX 2008 Annual Technical Conference on Annual Technical Conference, ATC 2008, pp. 321–334. USENIX Association, Berkeley (2008)
7. CA/Browser Forum. Guidelines for the issuance and management of extended validation certificates (March 2014), <https://cabforum.org/wp-content/uploads/EV-SSL-Certificate-Guidelines-Version-1.4.6.pdf>
8. Hoffman, P., Schlyter, J.: The DNS-based authentication of named entities (DANE) transport layer security (tls) protocol: TLSA. RFC 6698, RFC Editor (August 2012)
9. National Institute of Standards and Technology (NIST), <http://www.dnsops.gov/dnssec-perform.html>
10. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. RFC 6962, RFC Editor (June 2013)
11. Evans, C., Palmer, C., Sleevi, R.: Public key pinning extension for http. Internet-Draft draft-ietf-websec-key-pinning-08, IETF Secretariat (July 2013)
12. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management - part 1: General (revision 3). Technical Report NIST Special Publication 800-57, National Institute of Standards and Technology (March 2007)