TLS with Trustworthy Certificate Authorities

Kevin Walsh
College of the Holy Cross
Worcester, Massachusetts

***Abstract*—Cloud platforms can leverage Trusted Platform Modules to help provide assurance to clients that cloud-based web services are *trustworthy* and behave as expected. We discuss a variety of approaches to providing this assurance, and we implement one approach based on the concept of a trustworthy certificate authority. *TaoCA*, our prototype implementation, links cryptographic attestations from a cloud platform, including a Trusted Platform Module, with existing TLS-based authentication mechanisms. TaoCA is designed to enable certificate authorities, browser vendors, system administrators, and end users to define and enforce a range of trust policies for web services. Evaluation of the prototype implementation demonstrates the feasibility of the design, illustrates performance tradeoffs, and serves as an end-to-end, proof-of-concept evaluation of underlying trustworthy computing abstractions. The proposed approach can be deployed incrementally and provides new benefits while retaining compatibility with the existing public key infrastructure used for TLS.**

I. Introduction

Users rely on cloud-deployed web services for security-critical functionality, so users should seek assurance that these services act only in ways that do not violate their expectations. Often, user expectations are related directly to the data returned from a service. For example, we might expect a request to https://fb.me/react-0.14.3.js will produce a response containing a specific version of a popular javascript library. In other cases, users are interested not just in the content of a response, but in the behavior of a web service before or after it processes a request. For example, participants in an online poll might reasonably expect their votes to be counted in aggregate but kept secret from others. User of an online password generator would expect the returned passwords to be freshly generated and not stored or leaked by the service. Similarly, online password managers are expected to keep data confidential.

A wide variety of techniques can help ensure cloud-based web services behave, from homomorphic encryption or model checking, to software privilege separation or hardware-enforced isolation. But even if a web service employs such techniques and is, in fact, *trustworthy*—i.e. always behaves in the manner users expect—there is no general framework for providing end users assurance of this fact. TaoCA is intended to provide such a framework. Before examining the general case, it is instructive to consider some special cases addressed by existing mechanisms. A recent W3C proposal for subresource integrity [1] addresses the javascript library scenario mentioned previously by embedding an *integrity tag*, a list of cryptographic hashes, within an HTML document's links to certain external resources. After the linked resource is fetched, its actual hash is compared against the integrity tag. Only if a match is found is the resource considered trustworthy. Several parties are involved: a document author, who defines a trust policy; a guard within the client-side browser to enforce the policy; and web services to provide the linked resources. Here, policy expressiveness is limited as it is defined only in terms of response content, not behavior, and even then only exact matches are supported. This scheme also relies on protecting the integrity—typically using HTTPS—of the HTML-embedded trust policies as they are sent to the client.

HTTPS relies on TLS to provide users assurance about the identity of a web service. A browser allows a request intended for some domain D to proceed only after checking that the remote TLS endpoint speaks for D. In the common case, the browser relies on a public key infrastructure (PKI) to provide a binding between server TLS keys and domain names. Bindings are encoded in X.509 certificates issued by certificate authorities (CAs), where a binding is considered by the browser to be trustworthy if there is a corresponding chain of certificates leading back to a set of fixed trust anchors maintained by the browser. The PKI is a significant weakness in HTTPS-based assurance, since it relies in part on principals that have repeatedly been found to be untrustworthy [2], [3].

A second weakness in HTTPS-based assurance stems from the limited notion of *principal* used by CAs in practice: public keys and domain names, rather than details about the particular software or hardware underlying a web service. The later is directly relevant to evaluating whether a web service is trustworthy, since it is the software and hardware that most directly implement the behavior of the web service. A domain name alone will, at best, only reveal the identity of domain administrators, adding yet another level of indirection to the assurance HTTPS was meant to provide. Thus a user seeking assurance about web service behavior must now also seek assurance that administrators both safeguard the key and use properly-configured, trustworthy hardware and software.

We aim to provide more general assurance, beyond that provided by the above approaches: assurance that the service itself—its software and hardware implementation, relevant configuration details, and perhaps even its current execution state—satisfies some policy. For example, a policy might stipulate that the service be running a particular trusted implementation of an online voting scheme, with a known-good configuration, on trusted hardware.

Prior work has examined how to use the capabilities of a Trusted Platform Module (TPM) to provide assurance to clients. With some effort, TPM remote attestation can be leveraged to bind TLS endpoints to a specific platform configuration or identity [4]–[6]. Alternatively, the TPM can directly attest to the static and dynamic content delivered by a web server [7], in effect linking the content to a specific platform configuration. A cloud provider can also use TPM attestation to attest the source code underlying cloud-hosted services [8].

Any system using TPM attestation for web services must address three challenges. First, the performance of TPM hardware devices is extremely limited, so TPM operations must not

978-1-5090-3065-1/16/$31.00 ©2016 IEEE

be in the critical path. It is typically infeasible to involve the TPM during TLS handshaking or servicing web requests, for example. Second, the TPM's notion of principal is not directly compatible with X.509 certificate subjects. TPM principals comprise a public key associated with the TPM hardware device, together with a set of hashes that describe the platform. Third, information about the TPM principal executing at a remote endpoint must be securely conveyed to the end user, or more specifically, to user-trusted software that can evaluate the information and enforce policies on the user's behalf.

We propose to address these challenges by implementing a trustworthy CA on top of *Tao* [9], a TPM-enabled cloud computing platform for instantiating secure cloud services and attesting to properties of those services. Such a trustworthy CA would in turn produce certificates for web services running on other Tao platforms. We implemented *TaoCA* as a prototype to evaluate this approach. The prototype was further intended to help inform and evaluate the design of Tao itself. Thus, while TaoCA was built using Tao interfaces, several of the Tao features we describe in this paper are, in fact, a direct result of our experience building TaoCA. The TaoCA implementation described here is freely available[1] under an open-source license, as is Tao.[2]

TaoCA uses Tao to securely link TLS keys to the identity and configuration of the servers implementing a web service. TaoCA also conveys platform configuration details about web services to clients, so that browsers and other software controlled by end users can evaluate, and enforce policies about, the trustworthiness of those services. TaoCA only issues certificates to a server under circumstances that satisfy a certification policy specified in a formal logic. The logic allows CA administrators, end users, and other relying parties to specify and reason about cryptographic attestations—things principals *say*—and about trust relationships—when one principal can *speak for* another. Specifically, TaoCA can enforce policies based on Tao-attested and TPM-attested information about the principals requesting certificates. Moreover, because TaoCA itself runs on a TPM-enabled platform, information about TaoCA's own code, configuration, and state can also be conveyed to relying parties in a trustworthy manner.

The formal policies and cryptographic attestations used by TaoCA are all machine checkable, enabling relying parties to audit and verify not only the behavior of TaoCA, but also that of the web services certified by TaoCA. This work focuses on the problem of securely conveying to end users detailed information about the web services they use, e.g. software hashes, configuration data, and fingerprints of cryptographic keys. Ultimately, this information can be used to bootstrap enforcement of policies about higher-level properties, though specifying the form of such client-side policies is outside the scope of this paper. In combination, these approaches shift the nexus of trust away from certificate authorities and domain administrators, and towards relying parties, who can now enforce a richer set of client-specific certificate policies.

II. Attested CAs for Trustworthy Services

TaoCA functions much as a conventional CA, issuing certificates that bind the TLS keys used by HTTPS servers to identifying information about the principals holding those keys. Certificates convey this identifying information, eventually, to clients, so that end users can make trust decisions. A hierarchy of TaoCAs can function alongside conventional CAs, with some TaoCA instances serving as trust anchors and others acting as intermediate CAs. TaoCA can also support cross certification between CAs, though we do not discuss this for simplicity of presentation. We also omit discussion of certificate revocation, a complex problem for which TaoCA relies on existing, standard mechanisms. Figure 1 illustrates some of the principals involved in a simple scenario, highlighting some of the trust decisions these principals must make. For a multi-tiered web service, TaoCA would typically interact only with front-end servers that terminate HTTPS connections; the front-end servers would then be responsible for enforcing policies concerning the back-end servers they rely upon. In that case, TaoCA provides the means to assure users that such back-end security policies are in effect.

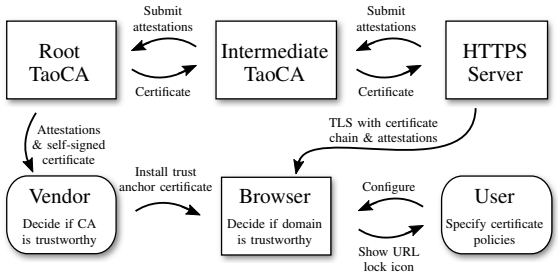

Fig. 1. A user invokes a browser to contact an HTTPS web server. The server holds a TLS key and matching X.509 certificate chain from an intermediate CA that is subordinate to a root CA, where the root CA is designated as a trust anchor by the user's browser vendor.

A. TaoCA Signing Key Management

Each TaoCA instance is associated with a certificate signing key. TaoCA protects the corresponding private key material from exposure by encrypting it using the facilities of a TPM-enabled platform. The hardware TPM then enforces a decryption policy that governs access to the raw key material, restricting access to only the specific software that implements TaoCA on a specific hardware platform. TaoCA in turn was designed to never expose the raw key material to other principals, not even to CA administrators.[3] An alternative, but less performant design could arrange for signing keys to be generated and stored entirely within the boundaries of the TPM device, with the TPM performing all key operations on behalf of TaoCA.

Whether the key material remains within a TPM or is exposed to the software implementing TaoCA, it is the TaoCA software that is responsible for processing certificate signing requests. These software components must be trusted to properly vet signing requests and enforce certificate policies. With software-generated keys, the software must also be trusted to safeguard the private key material. Thus the trusted computing base for TaoCA necessarily encompasses not just the TPM, but also the software for TaoCA, the underlying Tao platform, and all relevant configuration data.

[1] https://github.com/kevinawalsh/taoca
[2] https://github.com/jlmucb/cloudproxy

[3] Although TPM platforms are only nominally resistant to physical attacks, Tao could be adapted to use dedicated hardware security modules [10] or hardened platforms [11]–[13] to achieve better performance or provide greater assurance for storing raw key material.

In a conventional CA, some combination of operational procedures, hardware security modules, and software controls protects the private key [14]. Even so, administrators and other insiders typically have either direct access to the raw key material or privileges sufficient to invoke protected key APIs that use the key material, so those administrators must be trusted to not expose or misuse the key. This is in contrast to TaoCA, which is designed to function autonomously from local administration once it has been configured and initialized and before keys have been generated.

B. Domain Control and Attestation-Based Request Validation

A certificate authority issues a certificate in response to a *certificate signing request* (CSR) from some principal. Before issuing a certificate, TaoCA performs various checks to ensure the principal effectively controls the domains listed within the request. In keeping with common practice, TaoCA implements domain control validation strategies using a combination of email, DNS, and HTTP checks. These steps provide only the minimal level assurance provided by conventional CA domain-validated certificates. Email and HTTP validation, for example, are vulnerable to a variety of simple man-in-the-middle attacks.

Certificates issued by TaoCA are intended to provide additional, stronger assurances to relying parties beyond that provided by domain control validation or even conventional extended validation. For example, a TaoCA certificate can provide assurance that a web server's private TLS key is held by a specific version of software, in a particular configuration, on a known, trusted cloud platform. TaoCA can provide such assurances whenever the principal making (and named in) a certificate signing request is running on a compatible TPM-enabled, Tao cloud platform. In this case, TaoCA and the requesting principal execute a remote attestation protocol to exchange information about their platform configurations, using the TPM as a root of trust.

C. Certificate Policies and Certification Practices Statements

Each CA operates according to a *certificate policy* (CP), which defines the conditions under which that CA issues certificates. The CP details which domain control validation steps will be taken for different classes of CSR, for example. Each CA must also publish a *certification practices statement* (CPS) detailing these policies and the operating procedures governing the CA's behavior.

Certificates often include a *CPS pointer* URL that leads to a summary of the CA's CPS. TaoCA includes a CPS pointer in each certificate it issues. TaoCA also takes steps to ensure that the CPS is securely bound to certificates by including within each certificate a hash of the full CPS document. When a certificate is conveyed to some relying party—an end user or a browser vendor, for example—the linked CPS to is meant to help the relying party make a trust decision. By checking the hash included by TaoCA in the certificate, the relying party is assured of the integrity of the document obtained from the CPS pointer URL. This enables a client to meaningfully evaluate the suitability of the CPS. By contrast, for a conventional CA, the CPS pointer refers to a document that may change at any time, and these changes do not invalidate the certificate. In fact, it is recommended practice to make ex post facto changes whenever it is "judged by the policy administrator to have an insignificant effect on the acceptability of certificates", [15] because conventional CA administrators must be trusted in any case. But for a relying party, the CPS pointer in a conventional certificate is suspect, at best.

After examining a CPS and deciding whether it is suitable for some purpose, a relying party still needs assurance that the CA actually behaves accordingly. For conventional CAs, this assurance is gained mainly in an ad hoc fashion. For example, an independent auditor might manually certify CA practices. Certificate transparency [16] can also provide certain assurances, e.g. that the CA has not issued certificates for a domain except when requested by the domain's administrators.

TaoCA can offer stronger assurances about its behavior to relying parties, because TaoCA executes on a TPM-enabled cloud platform. Specifically, TaoCA publishes a TPM attestation that links its certificate signing key to the values of TPM platform configuration registers (PCRs). Relying parties can examine the attestation and PCR values to gain assurance that the hardware and software implementing the TaoCA instance is configured as documented in the published CPS. The TPM attestation will reveal, for example, that a particular TaoCA instance runs a specific version of the TaoCA software and has been configured to release a signed certificate to a requester only after posting the certificate to a public certificate transparency log. Similarly, the attestation includes configuration details specifying the number and types of domain control validation checks that TaoCA must perform before issuing a certificate, and it reveals whether the TaoCA has been configured to allow administrative intervention to manually approve or deny exceptional certificate signing requests. In effect, the CPS for a TaoCA instance becomes machine checkable, with the TPM attestation providing a proof of compliance.

The CPS and CP for a conventional CA are not machine checkable. Although they are typically developed according to a common framework [15], this framework is both complex and informal. Attempts have been made to formalize the structure and content of these documents [17], but they remain difficult to parse or evaluate.

D. Client Certificate Policies

When a client establishes a TLS connection to some web service, the client needs assurance that the web service will behave as expected by the client. To this end, TaoCA includes in each certificate information about the subject named in the certificate, along with details about the circumstances under which the certificate was issued. In particular, TaoCA can include PCR hashes describing the platform state, attested by TPM hardware, for the requesting principal's platform. For a simple HTTPS server, this includes complete details about the server software, its configuration, and the underlying cloud platform. For a more complex multi-tiered web service, the information included by TaoCA can extend to cover multiple servers. For example, if front-end servers that terminate HTTPS connections are configured to only communicate with specific back-end databases, caches, or payment gateways, and only over authenticated connections—as indeed a trustworthy web service would—then this configuration information would be included in TaoCA certificates for those front-end servers, along with a hash of the code that enforces such policies.

TaoCA certificate details can be used by clients to enforce custom certificate policies. In the simplest case, a client might

only accept certificates for HTTPS servers if the TaoCA certificate reveals that the servers execute on previously-vetted Tao cloud instances, for example, or that the servers execute known and trusted configurations of specific server software. More generally, TaoCA certificate details can be used to bootstrap higher level policies, e.g. that a payment server's software is included on a list of software approved by the user's bank.

By including details about subjects in certificates, TaoCA enables clients to side-step CA certificate policies, enforce their own client-specific certificate policies, and reduce the degree of trust placed in CAs. By contrast, clients that rely on certificates issued by a conventional CA have little opportunity to make trust decisions about the subjects of those certificates; certificate policies are instead dictated and enforced solely by the CAs. This is reflected in the certificates issued by a conventional CA, which include very little useful information—typically just a key, a domain name, and perhaps the name and address of a business.

E. TaoCA Certificate Chains

TaoCA can function either as a root CA or as a subordinate CA, and TaoCA can in turn issue certificates for subordinate CAs. When used as a root CA, a TaoCA instance must be included as a trust anchor before a client can use its certificates. Operating systems, browsers, and other clients maintain lists of trust anchors, provided and updated by vendors, perhaps with modifications made by the user or local system administrators. Typically, ad hoc mechanisms are used to maintain and update these lists [18].

The combination of TaoCA features described in the previous section facilitate a more systematic and less error-prone procedure for deciding whether a CA should be included as a trust anchor. A vendor can verify a TPM attestation to confirm the details of the CPS for that TaoCA instance, for example, and evaluate the included details about how that TaoCA issues certificates. Similarly, the extra information included by TaoCA within certificates, backed by independent TPM attestations from the subject's own platform, can provide a basis for auditing TaoCA behavior. Moreover, the degree of trust required for including a TaoCA instance as a trust anchor can, in principle, be lower than for a conventional CA. This is because TaoCA enables relying parties to directly implement their own certificate policies.

When one TaoCA instance issues a certificate for a subordinate TaoCA instance, information about the subordinate's platform is included in the certificate, just as would be the case for any other certificate subject. These platform attestations are signed by the TPM hardware underlying the subordinate TaoCA's cloud platform. Thus an X.509 certificate chain involving multiple TaoCA instances essentially includes two parallel sets of signatures: those on the X.509 certificates, and an embedded TPM signature on the platform attestations for each subject. The root certificate in such a chain is self-signed, so a relying party must independently check the signature for that certificate's TPM attestation. This signature check would normally be done by a vendor or administrator prior to adding the self-signed TaoCA certificate as a trust anchor. For intermediate and leaf TaoCA certificates, the issuing TaoCA checks the TPM signature before issuing the certificate. Thus standard X.509 certificate chain validation is sufficient to validate the platform attestations included in the intermediate and leaf certificates, without further signature checks. Alternatively, and departing from standard X.509 certificate chain handling, a relying party could re-verify the TPM signatures by duplicating the checks performed when the certificate was issued.

TaoCA core server, client	1222
TaoCA policy engines	6326
Tao host and hosted program library	10343
Logging & time-stamping server, client	226
RPC discovery server, client	445
Static file web service	220
Password web service	1060
Log viewer web service	221
Total	20063

TABLE I. APPROXIMATE LINES OF CODE

III. PROTOTYPE IMPLEMENTATION DETAILS

TaoCA runs on a Tao platform that has been modified and extended beyond what has been described in prior work [9]. In this section we detail both the TaoCA prototype implementation and the extensions we made to Tao. We also provide a brief overview of core Tao features.

A *Tao host* launches and manages *hosted processes*, which can take the form of simple Linux processes, docker containers, or CoreOS instances running within a KVM virtual machine. A Tao host provides a variety of trustworthy-computing services to TaoCA and other hosted processes, such as a CSPRNG, key generation and management, sealed storage for opaque secrets, and attestation services. The interface a Tao host exports is designed to be stackable, such that one Tao host instance can execute as a hosted process on a parent Tao host, with the parent Tao serving as a root of trust for the hosted Tao. Alternatively, Tao hosts can execute directly on a TPM-enabled platform and make use of the hardware TPM to provide a root of trust. Several Tao hosts can be managed together as a single *Tao domain*, so that a domain administrator can set domain-wide policies. These include a policy describing which programs are authorized to execute as hosted processes (or nested Tao hosts) within the domain, along with authentication and authorization policies to be used when the domain's hosted processes attempt to communicate with each other or with external principals.

Beyond providing support for TaoCA, Tao also serves as a platform for auxiliary TaoCA services, including an authenticated network logging and time-stamping service and an RPC discovery service. For testing, we also used Tao to prototype a variety of user-facing HTTPS services, including a basic web server for static files, a password generator and checking service, and a web-based log viewer for TaoCA's logging facility. Work is in progress on a simple secure voting service and a certificate transparency log service. TaoCA, Tao, and all of the services we implemented are written in Go and were deployed on Linux servers. Table I shows the approximate amount of code written for various components in our system.

TaoCA supports two modes of execution: manual or automated. In manual mode, TaoCA relies on an administrator to vet and approve certificate signing requests. In automated mode, TaoCA relies on a formal certificate policy, specified as an ACL or as a set of Datalog rules. In both execution

modes, TaoCA relies on Tao for generating and storing the certificate signing key. At no point is the raw key material exposed outside of the boundaries of TaoCA. This boundary includes the TaoCA code itself running as a Tao hosted process, parent Tao hosts, and so on, up to some root Tao host, along with the TPM and local platform hardware upon which this software depends for correctness.

A. TaoCA and Tao Principals

TaoCA makes extensive use of a formal logic, derived from Nexus Authorization Logic [19], for naming principals, describing the trust relationships between them, and reasoning about the attestations issued by principals. We integrated this logic into the underlying Tao platform by implementing *Tao principal names*, thereby enabling such names to be used within Tao, within TaoCA, and across all Tao-hosted services.

A Tao principal name is a unique identifier for an instance of a Tao host, Tao hosted process, cryptographic key, or any other principal that might authenticate to Tao components or be named in authorization policies. A simple, key-based principal name has the form $key(k)$. This name identifies the principal holding a particular private key, where k encodes the corresponding public key material and algorithm parameters. Key-based principals can sign statements in a credentials-based authorization logic, with the resulting attestation conveying a $says$ relationship within the logic. Formally, statement S, signed by key k, conveys that $key(k)\ says\ S$. This logic also supports a *speaksfor* relation between principals, such that from $(a\ says\ (b\ says\ S))$ and $(a\ speaksfor\ b)$ we can derive $(b\ says\ S)$. The *speaksfor* relation can be used as a form of delegation between independent principals.

TaoCA's authorization logic supports hierarchical *sub-principal* names for principals that do not necessarily hold unique private keys. For example, a Linux process hosted on a Tao instance may be identified as $key(k).Process(h)$. Here, the parent principal $key(k)$ is the name of the Tao host, uniquely identified by a public key k, and the extension $Process(h)$ includes a hash h of the program binary being executed. Tao includes additional details for its hosted processes, such as a process ID and a timestamp, but for brevity we omit these. Principals can also extend their own name with further details. Each TaoCA instance, for example, incorporates within its own principal name a hash of its CPS and a hash of its configuration files, which specify the operating mode—manual or automated—and certificate policy enforced by that TaoCA instance. A front-end web server might similarly extend its own name with a list of dynamically-loaded modules, or a summary of policies governing its interaction with back-end services.

A hosted processes can invoke its host Tao to obtain an attestation for a statement S. The host signs S on behalf of the hosted process, so the resulting attestation might convey, for example, $key(k)\ says\ (key(k).Process(h)\ says\ S)$. In the logic, parents speak for their sub-principals, so a recipient of this attestation can conclude $key(k).Process(h)\ says\ S$. This is particularly useful when the hosted process does not have its own private key.

In cases where a sub-principal has a public key pair, Tao supports a form of delegation using the *speaksfor* operator to bind those keys to the Tao principal name. When a TaoCA instance, represented by some sub-principal name T, generates a public key k_p, it requests an attestation from its host Tao to convey a delegation $T\ says\ (k_p\ speaksfor\ T)$. TaoCA can subsequently sign its own attestations using k_p because, coupled with the delegation, signing a statement S using k_p conveys the same information as having the parent Tao host sign S on behalf of the hosted TaoCA process.

Tao hosts can be stacked, so Tao sub-principal names are often several levels deep, with each level embedding details relevant to the behavior and security of that level:

$$key(k).CoreOS(h_c).Process(h_p).TaoCA(h_{cp}, h_{cps})$$

Here, a Tao host executes a CoreOS instance, which in turn executes a Linux process. The process functions as a TaoCA instance, enforcing a certificate policy and governed by a certification practices statement. A Tao host can also run directly on a TPM-enabled platform, with the TPM signing attestations on behalf of the Tao host, just as a Tao host can sign attestations for a hosted process. A TPM-based platform is identified using a Tao principal name of the form $tpm(k).PCRs(v_1, v_2, \ldots, h_1, h_2, \ldots)$. Here, k encodes the public attestation key for the TPM hardware, and each hash h_i specifies the value of TPM platform configuration register v_i. The PCR values identify the platform software and configuration at the time the attestation was generated. TaoCA and all related services execute on TPM-enabled platforms, so all are identified as sub-principals of a TPM.

B. TaoCA and Tao Authenticated Channels

Tao provides an authenticated, encrypted channel abstraction for hosted processes to communicate. For simplicity, Tao uses standard TLS protocols as the underlying transport mechanism to ensure confidentiality, but each TLS connection is established using self-signed certificates for both endpoints. The associated key pairs need not be stored long-term: a transient key can be generated on-demand for each connection or, to reduce the number of key generating and signing operations, a transient key can be reused for several connections. We extended this authenticated channel mechanisms to support TaoCA's authentication logic and its notion of principal by arranging for endpoints to exchange signed delegations at the start of each TLS connection. This authenticates each side of the connection using a Tao principal name.

As an example, consider a web server connecting to a TaoCA instance to obtain or renew an X.509 certificate. Suppose the web server executes as a Linux process with hash h_p, executing within a CoreOS container with hash h_c, executing on a TPM-enabled Tao host with TPM attestation key k and platform configuration registers x and y having values h_x and h_y. The web server first generates a transient TLS key k_p and a matching self-signed X.509 certificate, then uses these to establish a TLS connection with the TaoCA instance. Separately, the web server requests a delegation from its Tao host to convey:

$$k_p\ speaksfor\ tpm(k).PCRs(x, y, h_x, h_y).CoreOS(h_c).Process(h_p)$$

The web server sends the resulting signed delegation over the TLS connection to TaoCA. TaoCA in turn verifies the delegation signature using public key k, and it checks to ensure the value of k_p appearing within the delegation matches the value in the self-signed TLS certificate. Together, this authenticates the web server to TaoCA as the Tao principal name $tpm(k).PCRs(x, y, h_x, h_y).CoreOS(h_c).Process(h_p)$.

C. TaoCA Certificate Policies

TaoCA certificate policies are written in terms of the certificate details provided within certificate signing requests, the outcome of one or more domain control validation steps, and the Tao principal name p of the requesting process. Before approving a certificate signing request and issuing a certificate, TaoCA invokes an authorization guard to enforce the policy.

One type of guard implemented by TaoCA is based on ACLs. This guard is configured with a list of tuples of the form (p, s, d, v). The guard approves a certificate signing request from principal p for domain d, with X.509 subject details s only if (p, s, d, v) is among the ACL entries, where v is a subset of the keywords {"`http`", "`email`", "`dns`"} and all of the domain control validation steps in v have been successfully completed for this request.

The TaoCA prototype includes a second authorization guard that implements credentials-based authorization by querying a Datalog policy engine. The guard is configured with a list of Datalog rules, and it approves a certificate signing request from principal p for domain d and subject details s only if a predicate $Auth(\text{"}\texttt{ClaimCert}\text{"}, p, d, s, v)$ follows from the rules, where v again is the set of completed domain control validation steps.

Credentials-based authorization allows wide flexibility in choice of policy. For example, TaoCA might enforce the following two policy rules:

$$\begin{aligned} &\mathit{TrustedPrin}(P, D, S) \rightarrow \\ &\quad \mathit{Auth}(\text{"}\texttt{ClaimCert}\text{"}, P, D, S, \{\text{"}\texttt{email}\text{"}\}) \end{aligned} \tag{1}$$

$$\begin{aligned} &\mathit{TrustedPlatform}(H) \\ &\quad \wedge\ \mathit{TrustedHttps}(E) \wedge \mathit{Subprin}(P, H, E) \rightarrow \\ &\quad\quad \mathit{TrustedPrin}(P, \text{"}\texttt{foo.com}\text{"}, \text{"}\texttt{CN=Foo, Inc.}\text{"}) \end{aligned} \tag{2}$$

The first defines a *TrustedPrin* predicate and dictates that certificate signing requests from principals satisfying that predicate can be approved after undergoing email domain control validation. The second rule says that principals running *TrustedHttps* software on a *TrustedPlatform* satisfy the *TrustedPrin* predicate with the given domain name and X.509 subject details. The *TrustedHttps* and *TrustedPlatform* predicates would be defined by yet other Datalog policy rules. In the simplest case, for example:

$$\mathit{TrustedPlatform}(\mathit{tpm}(k).\mathit{PCRs}(x, y, h_x, h_y)) \tag{3}$$

$$\mathit{TrustedHttps}(\mathit{ext}.\mathit{CoreOS}(h_c).\mathit{Process}(h_p)) \tag{4}$$

Together, these rules would cause TaoCA to approve certain certificate signing requests—those specifying X.509 common name "Foo, Inc." with domain name "foo.com" and that have undergone email domain control validation—from a particular HTTPS server stack and platform.

Credentials-based and ACL-based guards can be extended to support policies that specify additional constraints, such as maximum validity periods or limitations on X.509 extensions.

IV. TaoCA Certificate Assurance

To provide greater assurance to relying parties, TaoCA publishes extensive information about the Tao principals to which it issues certificates, beyond that normally included in X.509 certificates. Specifically, TaoCA publishes with each certificate the Tao principal name p obtained from the authenticated Tao channel over which the certificate signing request was received. TaoCA also publishes the full certificate policy enforced by its guard, i.e. the full ACL or Datalog rule set.

TaoCA does not include all details directly within each X.509 certificate, as that would likely require changes to TLS clients. Additionally, these documents can be large and may not be needed by all relying parties. Instead, TaoCA embeds URLs in the CPS pointer and user notice extensions of each X.509 certificate. Relying parties can then fetch the documents as needed. To ensure integrity, TaoCA always includes a cryptographic hash as the last component of the URL. For example, the CPS pointer might specify:

$$\texttt{http://foo.com/ca\_cps/sha256.}h_{cps}\texttt{.txt}$$

where h_{cps} is the hash of the CPS document containing the Datalog policy rules. The web server's principal name and accompanying attestations would be linked with a similar URL in the user notice extension. The domain serving these documents need not be trusted to maintain their integrity.

We considered alternative mechanisms for security linking X.509 certificates to the policy and Tao principal documents generated by TaoCA. TaoCA could embed HTTPS URLs rather than HTTP URLs. This alone would be insufficient for assuring the integrity of the documents, as it requires relying party to establish trust in the HTTPS software serving the documents, and such trust is founded upon the same X.509 infrastructure in which TaoCA participates. Alternatively, subresource integrity [1] would provide a convenient mechanism for ensuring integrity. This is not currently defined for use in X.509 CPS pointer and user notice extensions, however. A third option employs a web proxy service to verify hashes. Here, the URL in the certificate extension might be:

$$\texttt{http://}h_{cps}\texttt{.sha.hash/foo.com/ca\_cps.txt}$$

A server for (hypothetical) domain `sha.hash` would fetch the actual URL, http://foo.com/ca_cps.txt, and validate the resulting document's hash against h_{cps} before returning the document to the client. Of course, a client would want assurance in the behavior of this web proxy service. Naturally, such assurance can be provided if the proxy itself is implemented on Tao and has a certificate from TaoCA. A less trusting client could instead simply implement such a service entirely locally.

When a relying party trusts a TaoCA instance, the CPS pointer and user notice help establish trust in the principal named in a certificate. To bootstrap trust in the TaoCA instance, a relying party needs assurance that the TaoCA instance executes on a suitable trusted platform and is configured with a suitable policy. When operating as a root CA, TaoCA publishes a self-signed certificate for its signing key. Included within that certificate is a link to its own certificate policy in the CPS pointer certificate extension, along with a link to its own Tao principal name and attestations in the user notice extension. These alone do not provide assurance, however, since the certificate is self-signed. But as mentioned previously, when a TaoCA instance begins execution it incorporates hashes of its CPS, certificate policy, and other configuration data into its Tao principal name. The TaoCA can then obtain a delegation from the host Tao linking its full principal name and its public certificate signing key. Before installing a TaoCA self-signed certificate as a trust anchor, a vendor or administrator can verify the Tao attestation to gain assurance that the certificate signing

key is actually held by a TaoCA installation configured as described by the CPS pointer and user notice extensions.

For a TaoCA instance subordinate to another TaoCA, the signed certificate for the subordinate includes its Tao principal name and associated attestations, signed by the Tao and TPM hardware for the subordinate, and verified by the parent TaoCA. Because the subordinate's principal name includes a hash of its own certificate policy and configuration, any party that trusts the parent TaoCA gains assurance about the certificate policy enforced by the subordinate as well. Thus a chain of certificate policies and Tao principal names extends from a leaf certificate for a web service, through zero or more intermediate TaoCA instances, up to some root TaoCA instance.

V. Performance

TaoCA performs more cryptographic operations in the course of validating a CSR and issuing a certificate than a conventional CA. And when a web service obtains a certificate from TaoCA, the web service and the Tao platform on which it runs must both perform various cryptographic operations to provide attestations required by TaoCA, beyond those needed when using a conventional CA. These cryptographic operations impose a cost for TaoCA and the web services using it. Note that existing clients of TaoCA-enabled web services are not required to perform any cryptographic operations beyond those needed for standard TLS, so we expect any direct impact to clients to be minimal. Specifically, the impact on clients depends primarily on the complexity of client-specific trust policies; we do not describe or measure such policies here.

To quantify potential performance impacts, we performed a series of experiments to compare the TaoCA prototype with the cfssl [20] PKI and TLS toolkit (version 1.1.0). We selected cfssl because it is popular—it is written and used by CloudFlare, a large content delivery network, for example, and it is used by the Let's Encrypt open source CA project—and because cfssl is written entirely in Go, like TaoCA. The shared implementation language allows a more direct comparison of the performance and designs of TaoCA and cfssl. All software was compiled using the same Go 1.4.2 compiler.

Both TaoCA and cfssl use the same underlying Go crypto libraries. Both servers acted as intermediate CAs, one level below a separate, offline root CA. Domain control validation was disabled for all experiments. All certificates and TLS connections use 256-bit ECDSA keys. The TPM used by TaoCA uses 2048-bit RSA keys for signing attestations. Experiments were carried out on a small cluster of Linux machines acting as load generators and one machine running a TaoCA or cfssl server, all equipped with TPM 1.2 chips. The server included dual quad-core 3.40 GHz Intel Haswell i7 processors, 8 GB of memory, and 1 GbE network interface. TaoCA and cfssl are primarily compute-bound services; when called for by the experiments, the modest computational resources of this server were easily saturated by the load generators.

A. CA Certificate Issuing Throughput

The performance impact of our approach can most readily be observed by measuring CA *throughput*, the rate at which the CA can validate certificate signing requests and issue certificates. For some CAs, even a very modest throughput is sufficient. For example, during it's first 31 days of public testing, the Let's Encrypt CA issued 204,435 certificates,[4] averaging about five certificates per minute. This CA recommends certificate renewal after 60 days; other CAs have more clients but typically much longer renewal intervals. By comparison, TaoCA certificates are tied to specific web service instances, so TaoCA may be expected to issue certificates much more frequently, perhaps each time a web service restarts or undergoes a security-relevant configuration change.

	throughput (certs/sec)	CPU utilization	e2e latency (ms) WAN	LAN
TaoCA	249.6	93.59%	542.5	26.57
cfssl/HTTP	1158.3	94.40%	212.2	5.44
cfssl/TLS	621.4	94.19%	428.1	13.50
cfssl/MTLS	365.3	97.08%	430.7	16.64

TABLE II. Performance of a single TaoCA or cfssl instance.

We measured the maximum throughput of TaoCA using open-loop load generators to submit increasing numbers of CSRs. We measured end-to-end latency separately, using a single closed-loop load generator, averaged over 1000 runs on local and emulated wide-area networks. As a baseline for comparison, we performed the same experiment for cfssl using a default HTTP-based API configuration. The results, summarized in the first rows of Table II, reveal that TaoCA can issue about 250 certificates per second, compared to almost 1160 per second for cfssl. In this experiment, the server CPU was nearly saturated for all configurations.

We attribute the reduction of observed TaoCA throughput to two causes: TaoCA accepts each CSR over a mutually-authenticated TLS connection; and before issuing a certificate, TaoCA verifies two attestations—one issued by the requester's TPM and a second issued by the requester's Tao host—and invokes the Datalog policy engine to enforce the certificate policy. By contrast, cfssl in its default configuration accepts CSRs over an unauthenticated HTTP connection, and it does no CSR validation. To quantify how these differences affect throughput, we ran experiments for two additional cfssl configurations: a TLS configuration in which each CSR is submitted to cfssl over an HTTPS-based API and requesters verify the server's API certificate; and a mutually-authenticated TLS configuration (MTLS), in which both requester and cfssl verify certificates when establishing a connection.[5] The results, shown in Table II, reveal that simply enabling TLS reduces cfssl throughput by 46%, and having cfssl validate a client TLS certificate for each request further reduces cfssl throughput to within a factor of 1.5 of the TaoCA prototype. We attribute the remaining difference in throughput to the cost of checking attestations and enforcing the Datalog certificate policy in TaoCA.

B. Certificate Issuing Latency

Traditionally, obtaining a signed certificate has not been part of the critical path for a web service. However, recent trends may change this. For example, many cloud platforms can automatically and very rapidly deploy and launch new

[4] http://letsencrypt.org/stats/

[5] Supporting MTLS required minor changes to cfssl. It may seem unusual to require that a requester use one X.509 certificate to establish a TLS connection in the course of submitting a request for another X.509 certificate. However, this situation is no less unusual than requiring a requester to use a secret "API key", a common practice supported by cfssl and other CA software.

server instances, and TaoCA could be called upon to issue a fresh certificate for each newly instantiated server. In such scenarios, low latency would be beneficial. Moreover, part of the end-to-end latency observed for obtaining a certificate stems from work done on the requester's platform, and this work differs between TaoCA and a conventional CA. With cfssl, the requester generates a key and matching CSR, then signs and submits the CSR. With TaoCA, before submitting a CSR, the requester must also obtain attestations from the underlying Tao host and TPM. To quantify these costs, we measured the end-to-end latency for a single closed-loop load generator to prepare and submit a certificate signing request then obtain a signed certificate in response. Both LAN and WAN configurations were tested, using a 1 GbE switched network with negligible delay and an emulated 128 MBit/s wide area network with 100ms average round trip delay.

In the wide area, end-to-end latency is dominated by round-trip time, as shown in the third column of Table II. Using HTTP, cfssl requires two round trips. TLS adds two more round trips to negotiate session keys and exchange certificates. The Tao-authenticated channels used by TaoCA also use TLS, but require one additional round trip to exchange Tao and TPM attestations. TaoCA WAN performance could be improved using TLS session resumption or TLS false start, or by embedding Tao and TLS attestations within TLS extensions.

Although a WAN is traditionally used to contact a CA, we envisage co-locating TaoCA instances and web services within the same Tao clouds. Indeed, Amazon has recently [21] begun moving to this CA deployment model for their cloud platform. Results for the LAN configuration, shown in the fourth column of Table II, indicate that end-to-end latency is low—well under 30 ms for all configurations tested—though much higher for TaoCA (26.57 ms) than for cfssl's default HTTP configuration (5.44 ms). The latency for cfssl with MTLS reveals that about 53% of the difference, 11.2 ms of 21.1 ms, can be attributed to TaoCA's use of mutually-authenticated TLS connections to submit CSRs. We attribute the remaining difference to two factors. First is TaoCA's verification of attestations and certificate policy enforcement. We measured this by instrumenting TaoCA and found this accounts for about 2.9 ms, or 14% of the added latency of TaoCA. The remaining 7.03 ms, or 33% of the added latency, is due to work performed by the requester to obtain attestations before submitting the CSR. In this experiment, the requester obtains one attestation from Tao and one from the TPM. The TPM attestation is generated only once, before the experiment begins, because it is bound only to the state of the Tao host and not the state of the web service executing as a Tao hosted process. The Tao attestation is generated fresh for each requester, however, and we include this cost in our results.

VI. Discussion and Related Work

TaoCA supports incremental deployment, as an unmodified TLS stack is sufficient to connect to TaoCA-certified web services. Legacy clients pay no direct cost for TaoCA, but all users benefit from increased CA transparency. Further, by examining TaoCA certificates and the embedded URLs within the CPS pointer and user notice fields, even users with TaoCA-oblivious clients can obtain new information to help judge the trustworthiness of the web services on which they rely. In our current prototype, users must manually download the documents linked by these URLs, verify their hashes, then inspect the included certificate policies and Tao principal names to decide if each TaoCA instance in the chain is trustworthy and, ultimately, whether the web service is trustworthy. We intend to automate this process by defining and enforcing formal, machine-checkable policies at the client. Our TaoCA prototype already provides much of the requisite implementation in the form of ACL and Datalog guards, and the information conveyed and attested by TaoCA provides the foundation to bootstrap higher-level client-side policies about web services.

Although we have focused on how a certificate authority can leverage TPM-based attestation to provide assurance to end users about the behavior of cloud-based services, prior systems have examined how cloud users and customers might directly leverage the TPM for the same purpose. Bouchenak et al. [22] review recent progress in this area. Spork [7] uses a TPM to attest to the content of responses from a cloud-based web server, allowing end users to verify the integrity of the server at the time the content was generated. That system also uses a novel batch signature scheme to reduce the cost of generating attestations. TaoCA avoids TPM overhead instead by performing attestation at the level of processes, rather than individual responses. Brown and Chase [8] perform attestation at the level of processes as well, relying on a trusted Eucalyptus cloud platform. That work side-steps the need for binding attestations to web service TLS keys by allowing clients to obtain attestations directly from the trusted Eucalyptus platform through an out-of-band channel. Eucalyptus and other cloud platforms have also been extended [23]–[26] to provide direct assurances to those deploying code or data to the cloud.

TaoCA relies on Tao for attestation and key management, and Tao in turn uses the facilities of the TPM, a device that is only nominally resistant to physical attacks. Anderson et al. [10] survey a variety of tamper-resistant hardware security modules (HSMs) that may offer higher levels of assurance. Recently, Amazon [27], Microsoft [28], and other providers have incorporated HSM devices into their cloud offerings. While an HSM can be useful for key management, these are special-purpose devices with rigid APIs and, in general, they are not suitable for directly executing network-facing services like TaoCA. Current HSM devices do not provide attestation for code hosted outside the HSM and, thus far, no public cloud provider offers attestation services for the processes they host.

Chhabra et al. [11] discuss hardening general purpose servers in a manner suitable for hosting and attesting TaoCA and web services. Haven [12] and MiniBox [13] rely on new SGX extensions for Intel CPUs to provide fine-grained attestation, isolation from cloud provider software, and protection against some physical attacks to platform hardware. Tao and TaoCA could use either to replace the TPM as a root of trust.

To facilitate efficient and meaningful attestation, services intended to be deployed to a trusted cloud need not be treated as black boxes. Twin Clouds [29] proposes the use of privilege separation to reduce the amount of code that needs to be attested, for example, while Lyle [30] uses a trusted cloud-based Java compiler and model checker to attest to properties of web services running in the cloud.

TaoCA requires few changes to clients because it links attestations to standard X.509 certificates and browsers can still use standard TLS and X.509 algorithms. Prior work [4], [5] modifies or extends TLS to integrate TPM-based endpoint attestation. Stumpf [6] analyzes a number of such protocols.

TaoCA leverages cloud platform support for CA and web services attestation. Some recent efforts focus instead on client-side validation of web service and CA behavior. Strict transport security and public key pinning [31] counteract attacks that strip or substitute rogue TLS credentials in HTTPS connections. Subresource integrity [1] allows a client to directly measure the integrity of resources linked from a trusted document. Certificate transparency [16] defines a mechanism for clients and web service administrators to audit the behavior of certificate authorities. DANE [32] and DNS Certification Authority Authorization [33] completely or partly eliminate the need for clients to trust the conventional TLS CA infrastructure, instead locating trust in DNSSEC. These client-focused approaches can work in concert with the approach we have taken in TaoCA.

VII. Conclusion

We propose the use of TPM-enabled cloud platforms to implement certificate authorities for cloud-based web services. To demonstrate the feasibility of our approach we implemented TaoCA, a prototype CA for the Tao cloud computing platform. Both TaoCA and the web services using it are attested by Tao, with a root of trust located in underlying Trusted Platform Modules. The combination of attested CAs and attested web services provides enhanced assurance to end users. Moreover, it enables clients to systematically enforce their own trust policies beyond what is possible with conventional certificate authorities.

VIII. Acknowledgments

Tom Roeder, John Manferdelli, and Fred B. Schneider collaborated extensively on the design and implementation of the TaoCA prototype and related modifications to Tao.

References

[1] D. Akhawe, F. Braun, F. Marier, and J. Weinberger, "Subresource integrity: W3C candidate recommendation," https://www.w3.org/TR/SRI/, Nov. 2015.

[2] S. B. Roosa and S. Schultze, "Trust darknet: Control and compromise in the internet's certificate authority model," *IEEE Internet Comput.*, vol. 17, no. 3, pp. 18–25, Feb. 2013.

[3] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Proc. Internet Measurement Conf.*, Oct. 2013.

[4] K. Goldman, R. Perez, and R. Sailer, "Linking remote attestation to secure tunnel endpoints," in *Proc. ACM Workshop Secure Web Services*, Nov. 2006.

[5] F. Armknecht, Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, G. Ramunno, and D. Vernizzi, "An efficient implementation of trusted channels based on OpenSSL," in *Proc. ACM Workshop Scalable Trusted Comput.*, Oct. 2008.

[6] F. Stumpf, "Leveraging attestation techniques for trust establishment in distributed systems," Ph.D. dissertation, Technische Universität Darmstadt, Darmstadt, Germany, 2010.

[7] T. Moyer, K. Butler, J. Schiffman, P. McDaniel, and T. Jaeger, "Scalable web content attestation," *IEEE Computer*, vol. 61, no. 5, pp. 686–699, May 2012.

[8] A. Brown and J. S. Chase, "Trusted platform-as-a-service: a foundation for trustworthy cloud-hosted applications," in *Proc. ACM Cloud Comput. Security Workshop*, Oct. 2011.

[9] J. Manferdelli, T. Roeder, and F. Schneider, "The CloudProxy Tao for trusted computing," University of Califronia at Berkeley, Berkeley, California, USA, Tech. Rep. UCB/EECS-2013-135, Jul. 2013.

[10] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors—a survey," *Proc. IEEE*, vol. 94, no. 2, pp. 357–369, Feb. 2006.

[11] S. Chhabra, Y. Solihin, R. Lal, and M. Hoekstra, "An analysis of secure processor architectures," *Trans. on Computational Science VII*, pp. 101–121, 2010.

[12] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proc. Symp. Operating Syst. Design & Implementation*, Oct. 2014.

[13] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry, "Minibox: A two-way sandbox for x86 native code," in *Proc. Annu. Tech. Conf.*, Jun. 2014.

[14] J. E. Martina, T. C. S. de Souza, and R. F. Custodio, "OpenHSM: An open key life cycle protocol for public key infrastructure's hardware security modules," in *European PKI Workshop*, ser. Lecture Notes in Comput. Sci., J. Lopez, P. Samarati, and J. L. Ferrer, Eds., vol. 4582. Berlin, Germany: Springer, Jun. 2007, pp. 220–235.

[15] S. Chokhani, W. Ford, R. V. Sabett, and C. R. Merrill, "Internet X.509 public key infrastructure certificate policy and certification practices framework," Internet Eng. Task Force RFC 3647, 2003.

[16] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," Internet Eng. Task Force RFC 6962, 2013.

[17] G. A. Weaver, S. Rea, and S. W. Smith, "A computational framework for certificate policy operations," in *European PKI Workshop*, ser. Lecture Notes in Comput. Sci., F. Martinelli and B. Preneel, Eds., vol. 6391. Berlin, Germany: Springer, 2010, pp. 17–33.

[18] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," in *Proc. IEEE Symp. Security and Privacy*, May 2013.

[19] F. B. Schneider, K. Walsh, and E. G. Sirer, "Nexus authorization logic (NAL): Design rationale and applications," *ACM Trans. Inf. and Syst. Security*, vol. 14, no. 1, pp. 8:1–8:28, May 2011.

[20] Cloudflare, Inc., "CFSSL: CloudFlare's PKI/TLS toolkit," https://github.com/cloudflare/cfssl, 2015.

[21] Amazon.com, Inc., "AWS certificate manager," https://aws.amazon.com/certificate-manager/, 2015.

[22] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, "Verifying cloud services: Present and future," *ACM Operating Syst. Review*, vol. 47, no. 2, pp. 6–9, Jul. 2013.

[23] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proc. ACM Cloud Comput. Security Workshop*, Oct. 2010.

[24] I. Khan, H. ur Rehman, and Z. Anwar, "Design and deployment of a trusted Eucalyptus cloud," in *Proc. IEEE Int. Conf. Cloud Comput.*, Jul. 2011.

[25] F. J. Krautheim, "Private virtual infrastructure for cloud computing," in *Proc. Hot Topics in Cloud Comput.*, Jun. 2009.

[26] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. Hot Topics in Cloud Comput.*, Jun. 2009.

[27] Amazon.com, Inc., "AWS CloudHSM," https://aws.amazon.com/cloudhsm/, 2013.

[28] Microsoft Corp., "Azure key vault—making the cloud safer," http://blogs.technet.com/b/kv/archive/2015/01/08/azure-key-vault-making-the-cloud-safer.aspx, 2015.

[29] S. Bugiel, S. Nürnbergr, A.-R. Sadeghi, and T. Schneider, "Twin clouds: Secure cloud computing with low latency," in *Commun. and Multimedia Security*, ser. Lecture Notes in Comput. Sci., B. D. Decker, J. Lapon, V. Naessens, and A. Uhl, Eds., vol. 7025. Berlin, Germany: Springer, 2011, pp. 32–44.

[30] J. Lyle, "Trustable remote verification of web services," in *Trusted Comput.*, ser. Lecture Notes in Comput. Sci., L. Chen, C. J. Mitchell, and A. Martin, Eds., vol. 5471. Berlin, Germany: Springer, Apr. 2009, pp. 153–168.

[31] M. Kranch and J. Bonneau, "Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning," in *Proc. Annu. Netw. and Distrib. Syst. Security Symp.*, Feb. 2015.

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

[33] P. Hallam-Baker and R. Stradling, "DNS certification authority authorization (CAA) resource record," Internet Eng. Task Force RFC 6844, 2013.