Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Parties

Italo Dacosta, Mustaque Ahamad, and Patrick Traynor

Converging Infrastructure Security (CISEC) Laboratory
Georgia Tech Information Security Center (GTISC)
Georgia Institute of Technology
{idacosta,mustaq,traynor}@cc.gatech.edu

Abstract. The security guarantees provided by SSL/TLS depend on the correct authentication of servers through certificates signed by a trusted authority. However, as recent incidents have demonstrated, trust in these authorities is not well placed. Increasingly, certificate authorities (by coercion or compromise) have been creating forged certificates for a range of adversaries, allowing seemingly secure communications to be intercepted via man-in-the-middle (MITM) attacks. A variety of solutions have been proposed, but their complexity and deployment costs have hindered their adoption. In this paper, we propose Direct Validation of Certificates (DVCert), a novel protocol that, instead of relying on third-parties for certificate validation, allows domains to *directly and securely vouch* for their certificates using previously established user authentication credentials. By relying on a robust cryptographic construction, this relatively simple means of enhancing server identity validation is not only efficient and comparatively easy to deploy, but it also solves other limitations of third-party solutions. Our extensive experimental analysis in both desktop and mobile platforms shows that DVCert transactions require little computation time on the server (e.g., less than 1 ms) and are unlikely to degrade server performance or user experience. In short, we provide a robust and practical mechanism to enhance server authentication and protect web applications from MITM attacks against SSL/TLS.

1 Introduction

The Secure Sockets Layer (SSL) protocol and its successor, Transport Layer Security (TLS), have become the de facto means of providing strong cryptographic protection for network traffic. Their near universal integration with web browsers arguably makes them the most visible pieces of security infrastructure for average users. While vulnerabilities are occasionally found in specific implementations, SSL/TLS are widely viewed as robust means of providing confidentiality, integrity and server authentication. However, these guarantees are built on tenuous assumptions about the ability to authenticate the server-side of a transaction by using digital certificates signed by a *trusted* third-party certification authority (CA).

S. Foresti, M. Yung, and F. Martinelli (Eds.): ESORICS 2012, LNCS 7459, pp. 199–216, 2012.

The security community has long been critical of the Public Key Infrastructure for X.509 (PKIX) and its CA-based trust model [13,19]. Much of the concern has focused on the role of the CAs and their ability and motivation to not only correctly verify and attest the coupling between an identity and a public key, but also to protect their own resources. Browsers and operating systems determine what CAs users should trust by default (i.e., trust anchors). However, this model has resulted in *hundreds of CAs, all equally trusted and from more than 50 different countries* [11]. Due to this excessive trust, CAs can forge certificates for any domain that will be accepted as valid by most browsers. Thus, adversaries can obtain forged certificates by coercing or compromising any CA and use them to execute man-in-the-middle (MITM) attacks against SSL/TLS connections. Last year, the number of reported attacks against CAs increased considerably [18, 22, 23, 34]. In some cases, adversaries were able to forge certificates for important web domains (e.g., google.com, yahoo.com and live.com). Even worse, it has been estimated that a forged certificate was used to intercept close to 300,000 Gmail sessions in Iran [26]. Furthermore, there is evidence that governments and private organizations are using forged certificates as part of their surveillance and censorship efforts [27,35,36]. The frequency of these incidents is likely to increase in the future, as more and more web applications rely on SSL/TLS to protect all their communications.

Multiple solutions have been proposed to deal with the threat imposed by forged certificates and MITM attacks. The most popular approach is the use of additional third-parties to extend or replace the rigid CA trust model (e.g., network notaries [30,38], public audit logs [12,25] and secure DNS (DNSSEC) [20]). In this approach, users can select one or more third-parties to vouch for the authenticity of a certificate, improving the chances of detecting a MITM attack. However, depending only on third-parties for certificate validation has several shortcomings such as: significant deployment and operational costs (e.g., additional infrastructure with high availability requirements), more complex trust model for users, privacy concerns and more complex revocation procedures. Therefore, *the inherent complexity and costs associated with third-party solutions have prevented their widespread deployment.* As a result, most users still rely on weak certificate validation checks to detect MITM attacks.

In this paper we propose Direct Validation of Certificates (DVCert), an efficient and easy to deploy protocol that provides stronger certificate validation and effective detection of MITM attacks without using third-parties. Our mechanism comes from a simple observation – users have already established secrets (e.g., passwords) with their most important web applications. *DVCert allows web applications to use these secrets to directly and securely attest for the authenticity of their certificates without exposing those secrets to offline attacks.* After a single round-trip DVCert transaction, a browser receives the information required to validate all the certificates that could be used during a session with the web application, including certificates from other domains. As a result, to execute a MITM attack, an adversary not only needs to compromise a CA but also each targeted web domain. A DVCert transaction uses a modified Password

Authenticated Key Exchange (PAKE) protocol known as PAK [8,28]. However, we are not simply applying a known protocol; rather, we modified PAK to provide *only* server authentication and integrity protection instead of mutual authentication and generation of encryption keys (i.e., traditional use of PAKE protocols). These changes allow better performance and simplify deployment without affecting PAK's formal security proofs. Our experimental evaluation shows that an optimized DVCert transaction requires little computation time on the server (e.g., < 1 ms) and on the browser. More importantly, DVCert transactions are executed at most once per session; thus, their impact on server performance or user experience is negligible. DVCert's design also provides multiple advantages over third-party solutions: simpler trust model, lower deployment and operational costs (e.g., no additional infrastructure is required) and no privacy risks. Finally, DVCert is a readily available mechanism designed to improve the current CA trust model and be compatible with third-party solutions such as DNSSEC, once these solutions are deployed in the future. In so doing, we make the following contributions:

– **Designing and implementing an efficient and easy to deploy mechanism to detect MITM attacks against SSL/TLS without third-parties:** We develop a protocol that provides more robust certificate validation and detects MITM attacks, even if the adversary uses forged certificates. By allowing web applications to attest directly for their certificates, our mechanism avoids many of the challenges hindering the deployment of third-party solutions. We implemented a proof-of-concept extension for Firefox and Firefox for mobile browsers and a PHP-based server component to demonstrate the deployability of our solution.

– **Conducting an extensive performance analysis in multiple platforms:** We characterize DVCert's performance using our prototype implementation in both desktop and mobile browsers. Our results show that an optimized DVCert transaction requires 0.54 ms of computation time on the server and 12.03 and 97.70 ms on a laptop and on a smartphone respectively. Compared to a naïve implementation, these results represent a 94.96%, 55.07% and 77.82% improvement on the server, laptop and smartphone correspondingly. Furthermore, we apply ProVerif [6] to formally verify DVCert's resilience to offline attacks.

– **Making our DVCert implementation available to the community:** The DVCert extension for Firefox and Firefox for mobile as well as the server PHP code are available for evaluation at: `http://www.cc.gatech.edu/~idacosta/dvcert/index.html`.

The remainder of this paper is organized as follows: Section 2 offers important background information on SSL/TLS and MITM attacks and presents our motivation; Section 3 provides the design and formal description of DVCert; Section 4 presents our security analysis of DVCert; Section 5 shows our experimental analysis and results; Section 6 offers additional analysis and discussion of our proposed protocol; Section 7 provides an overview of important related work; and Section 8 presents our conclusions.

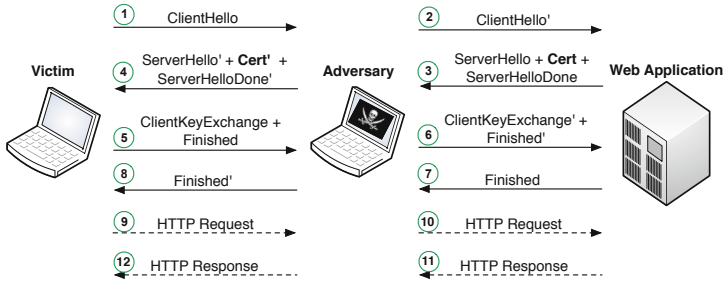

Fig. 1. Example of a MITM attack against SSL/TLS

2 Background and Motivation

2.1 The SSL/TLS Protocols and Web Applications

The SSL/TLS protocols [10, 17] are the main security mechanisms used to protect the communications between browsers and web applications. By providing a transparent encryption layer, SSL/TLS guarantee the confidentiality and integrity of the data traveling across the Internet. Moreover, SSL/TLS allow browsers to authenticate web application's servers via X.509 digital certificates [2]. A digital certificate binds the server's identity (i.e., domain name) to the server's public key and it is signed by a Certification Authority (CA) trusted by both the server and the browser. Initially, due to performance considerations, most web applications used SSL/TLS only to protect requests carrying private data (e.g., passwords, credit card numbers). However, due to the increasing number of attacks against web sessions (e.g., session hijacking), many applications have been forced to protect all their communications with SSL/TLS. For this reason, is common that during a session, a browser establishes multiple SSL/TLS connections not only with web application's servers but also with servers from third-party domains (e.g., CDNs and ads networks). Through a short survey from the Alexa Top 20 US sites and popular online banking sites (15 in total), we determined that an average of 12 certificates per domain were validated by the browser, with a minimum of 4 and a maximum of 22. Moreover, most sites included at least one certificate from a third-party domain.

2.2 MITM Attacks against SSL/TLS

The security guarantees offered by SSL/TLS rely on the correct authentication of the server. All such guarantees are rendered ineffective if an adversary is able to convince users to accept an illegitimately generated certificate, as shown in Figure 1. First, the adversary positions herself in the network path between the victim's computer and the server. When the victim sends a request for establishing a new SSL/TLS connection with the server (message 1), the adversary intercepts and responds to it (message 4) using a forged certificate (Cert'). If the

victim accepts this certificate, then she completes the SSL/TLS setup with the adversary (messages 5 and 8), who has, as a result, successfully masqueraded as the server. Simultaneously, the adversary establishes a new SSL/TLS connection with the server (messages 2, 3, 6, and 7). At this point, the adversary has two active SSL/TLS connections: one with the victim and one with the server. However, from the victim's and server's perspectives, there is only one secure connection in place. The adversary can now decrypt, re-encrypt and forward all the messages exchanged between the victim and the server (messages 9 to 12). As a result, the adversary can access private information (e.g., passwords) or even modify it (e.g., code injection).

2.3 Problems with Third-Party Solutions

A considerable number of mechanisms have been proposed to improve server-side authentication and protect against MITM attacks (see Section 7). The most popular approach is the use of additional third-party entities that can also vouch for the authenticity of server certificates. Third-party solutions provide a number of benefits: protection of the first connection to a new domain, scalable attestation of certificates for all public domains and minimal requirements for web applications. Unfortunately, this approach also faces several critical challenges. First, *these mechanisms have significant deployment and operational costs*. The additional infrastructure needed can be expensive to deploy and operate due to requirements such as high-availability, data consistency, performance and security. Even web applications can be affected by the operational overheads required by these mechanisms. Second, *the resulting trust model is more complex*. The use of multiple trusted entities to choose from can make the trust model more complex to evaluate and understand. Thus, average users are likely to rely on default trust configurations. Moreover, trust is dynamic – a trusted entity today may become an adversary tomorrow. Third, *these mechanisms introduce new privacy risks*. Users' browsing activity is disclosed to third-party entities. Preventing this problem can add complexity to these solutions. Fourth, *certificate revocation procedures become more complex*. The use of multiple entities make revocation more difficult because of the additional overhead required to revoke multiple proofs of authenticity (e.g., signatures). Finally, *captive portals typically interfere with these mechanisms*. In places such as airports and hotels, captive portals can block requests for certificate validation to external entities before user registration. Thus, captive portals need to be modified to allow additional certificate validation mechanisms.

3 Direct Validation of SSL/TLS Certificates

We present Direct Validation of SSL/TLS Certificates (DVCert), an efficient and practical mechanism that improves certificate validation and provides stronger protection against MITM attacks. Instead of relying on third-parties, DVCert uses the existing shared secrets between the user and the web application to

directly validate server certificates. DVCert overcomes the limitations of third-party solutions while also reducing the risks associated with using low-entropy keys in network protocols.

3.1 Scenario and Threat Model

Our scenario assumes a large, highly distributed web application. The application uses SSL/TLS to protect all the communications with its users (i.e., always-on HTTPS). To establish SSL/TLS connections, the application has multiple certificates signed by a trusted CA. In addition, the application's web pages include content from third-party servers. These servers also communicate using SSL/TLS and have their own valid certificates. We assume that SSL/TLS are correctly configured in the application's servers as well as in the third-party servers. Furthermore, users share a password with the application and use HTML forms for authentication. Instead of plaintext passwords, the application stores password salted hashes using public salt values. Finally, we assume that users follow a robust password policy that is enforced by the application.

We consider a polynomial-time (PPT) adversary that has access to all the communication between the web application and its users. The adversary's goal is to eavesdrop and tamper with this communication by executing MITM attacks against SSL/TLS. To perform such attacks, we assume that it is possible for the adversary to obtain forged certificates for any domain that are signed by some trusted CA. However, the adversary does not have access to users' passwords, password salted hashes or server's private keys. Moreover, we do not consider attacks against user computers or application servers to obtain such information and attacks that exploit SSL/TLS implementation or configuration errors.

3.2 Desired Protocol Properties

We identified properties required to achieve an effective and practical defense against MITM attacks and use them to design DVCert. (1) *Effective detection of MITM attacks*: the proposed mechanism must provide robust server authentication and effective detection of MITM attacks against SSL/TLS, even if illegitimately obtained certificates are used. (2) *Robustness against offline attacks*: the proposed mechanism should not leak information about the user's authentication credentials and must be resilient to offline attacks such as dictionary and cryptanalytic attacks. (3) *Deployability*: the proposed mechanism should not require additional hardware or software, only small changes to the browser and web application. In addition, it should be simple to configure in both the browser and the web application. (4) *Performance*: the proposed mechanism must be efficient. It must not affect the overall performance and scalability of the web application. Moreover, it should not introduce risks of DoS attacks. (5) *Privacy*: the proposed mechanism should not disclose user information to third-parties and adversaries. (6) *Compatibility*: the proposed mechanism must not interfere with existing functionality in the browser and web application. Browsers not

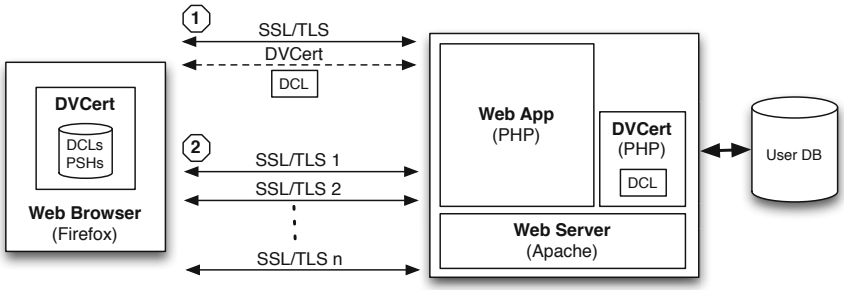

Fig. 2. High level overview of DVCert. (1) The browser uses a DVCert transaction to obtain a fresh DCL (Domain Certificate List); (2) it uses the DCL to validate certificates used in all the SSL/TLS connections with the application.

supporting the proposed mechanism should still be able to access the web application. Moreover, the proposed mechanism must be compatible with other certificate validation protocols. (7) *Usability*: the proposed mechanism should require minimal user intervention and have minimal impact on user experience. (8) *Simple trust model*: the proposed mechanism should have an easier to understand trust model in comparison to third-party solutions. Users must not be required to make additional trust assessments.

3.3 Protocol Description

MITM attacks against SSL/TLS connections are possible because server certificates are validated using only a single third-party signature and mutual authentication is weak. DVCert addresses these problems by allowing web applications to use already available shared secrets to *vouch directly* for the authenticity of certificates instead of relying only on third-parties. Figure 2 shows a high level description of the DVCert protocol. First, the browser establishes a SSL/TLS connection with the web application and then executes a DVCert transaction based on the user's password and a modified PAKE protocol (step 1). In this transaction, the browser authenticates the web application and receives its latest certificate information. The certificate information is shared using a Domain Certificate List (DCL), a data structure maintained by the web application that contains the *fingerprints*[1] *of all the certificates* that could be used during a session with the application. The DCL not only includes the fingerprints of the application's certificates but also of third-party's certificates used in the application (e.g., CDNs and ads networks). Second, the browser stores the DCL temporarily and uses it to validate the certificates of each SSL/TLS connection with the application (step 2), including the SSL/TLS channel established in step 1. If a certificate is not found in the DCL, then the corresponding SSL/TLS connection is flagged as untrusted (i.e., probable MITM attack). Once the DCL expires, a

[1] A certificate fingerprint is the cryptographic hash of the binary representation (e.g., DER encoding) of the certificate.

Shared information: g, p, $d = domain$, $s = H(u|d)$. Hash functions H, H_1, H_2, H_3, H_4
Information held by Browser: $u = username$, $pw = password$
Information held by Server: $P = H(pw|s)$, $DCL = domain\ certificate\ list$

Browser		**Server**
$a \in Z_q$ $P = H(pw\|s)$		
$m_1 = g^a \times H_1(u\|d\|P) (\mathrm{mod}\ p)$ (1)	$\xrightarrow{u,\ m_1}$	$m_1 \bmod p \overset{?}{\neq} 0$ $b \in Z_q$ $g^{ab} = (\frac{m_1}{H_1(u\|d\|P)})^b (\mathrm{mod}\ p)$ $m_2 = g^b \times H_2(u\|d\|P) (\mathrm{mod}\ p)$ $r = (u\|d\|P\|g^a\|g^b\|g^{ab})$ $h_1 = H_3(r\|H(DCL))$
$g^{ab} = (\frac{m_2}{H_2(u\|d\|P)})^a (\mathrm{mod}\ p)$ $r = (u\|d\|P\|g^a\|g^b\|g^{ab})$ $h_1 \overset{?}{=} H_3(r\|H(DCL))$ $h_2 \overset{?}{=} H_4(r)$	(2) $\xleftarrow{m_2,\ h_1,\ h_2,\ DCL}$	$h_2 = H_4(r)$

Operations:
$x|y$: concatenation of strings x and y
$H^i(x)$: $i\text{-}th$ standard cryptographic hash of x
$H_i(x)$: special agreed-on cryptographic hash of x [9, 21]

Fig. 3. Detailed description of a DVCert transaction. On each transaction, the server is authenticated and the browser securely receives a new DCL.

new DVCert transaction is executed (step 1) to update it. Finally, to avoid asking for the user's password on each transaction, the browser securely stores the password salted hash (PSH) together with the DCL.

DVCert achieves our goals by building on a significantly modified version of PAK [8, 9, 21, 28]. PAK (and the PAKE family of protocols) is based on the Diffie-Hellman (DH) key exchange and allows the use of low entropy secrets such as passwords to securely establish a session secret (i.e., authenticated Diffie-Hellman). PAK was selected as a starting point for our work because of its formal security proof and its ability to use shorter exponents [29] for better performance when compared to other related PAKE-based protocols. The major difference in our approach is that DVCert uses PAK *only* for server authentication instead of mutual authentication and generation of encryption keys (standard use of PAKE protocols), and include features to protect the integrity of the DCL and distinguish between tampering of the DCL and password errors. In other words, only the browser verifies the session secret established during the transaction. By not providing user authentication, DVCert requires fewer messages and, more importantly, avoids changes to the browser login user interface – a major challenge for the deployment of PAKE protocols in web applications [15]. Hence, DVCert is compatible with current user authentication mechanisms (e.g., HTML form-based authentication).

Figure 3 shows the details of a DVCert transaction (step 1 on Figure 2). First, the browser establishes a SSL/TLS connection with the server. This connection is used to protect protocol information (e.g., usernames) from eavesdroppers. Next, the browser generates a random exponent a (browser's DH secret), computes the

DH value g^a and uses it and the password salted hash P to compute m_1. If the password salted hash is not available for this domain (e.g., first DVCert transaction with this domain), then the browser prompts the user for her username u and password pw, computes the password salted hash P and stores it in a secure location for future transactions (i.e., the user is prompted only once for her password). Once m_1 has been calculated, the browser sends it and the username u to the server using a special header field in a HTTP request (message 1) over SSL/TLS. After receiving the DVCert request, the server verifies that $m_1 \neq 0$ to prevent a known attack, uses the username u to retrieve the password salted hash P from the server's database, generates the random exponent b (server's DH secret) and computes the DH value g^b. The server now obtains the browser's DH value g^a from m_1, calculates the session secret g^{ab} and computes m_2 and h_2. In addition, the server uses the latest version of the *DCL* to compute h_1. Next, the server sends m_2, h_1, h_2 and the *DCL* to the browser in the HTTP response (message 2). Then, the browser uses the received values to obtain the server's DH value g^b and to calculate the session secret g^{ab}. Next, the browser uses the session secret g^{ab} and other protocol state information to compute new h_1 and h_2 values. The browser now compares the computed h_1 with the one received from the server. If the values match, then the DVCert transaction was successful. Thus, the DCL file is trusted (i.e., has not been tampered with) and can be used to validate certificates. In addition, the successful verification of h_1 also proves the server's identity. If the h_1 values do not match, then the browser proceeds to verify h_2. If this verification succeeds, then the DCL has been modified and there is a high probability that a MITM is in progress. Therefore, neither the DCL nor any communication with the server can be trusted. The browser displays a warning to the user and halts the communications with the server. If the h_2 values are different, then the transaction could have failed due to a password error (e.g., user typed the wrong password) or a MITM attack. Thus, the browser displays a warning and prompts the user for a new password for a limited number of attempts. If the protocol still fails after several attempts, then the browser halts all communications with the server. In other words, h_2 is used to differentiate between protocol failures due to a MITM attacks or due to password errors.

After a successful DVCert transaction, the browser stores the DCL and the password salted hashes in a secure location isolated from other browser components. The browser stores one DCL per domain for a limited period of time according to a domain policy (e.g., once per session). Thus, the total number of DVCert requests per user is significantly lower than the total number of SSL/TLS connections. When a SSL/TLS connection is established with a server, the browser checks that the certificate is in the corresponding DCL (step 2 in Figure 2). If the certificate is not in the DCL, then a MITM attacks is likely to be in progress. Thus, the browser displays a warning to the user and halts the communications with the server. Once a DCL expires, the browser sends an automatic request (i.e., no user intervention) for a new DVCert transaction to update the DCL.

Finally, DVCert assumes that PAK constants, the prime number p and the generator g, are publicly known. For example, they can be hardcoded in DVCert's browser and server components. This measure is important to prevent an adversary from sending bogus p and g values and tricking the user into an improper DVCert exchange that could leak password information. Moreover, DVCert assumes that the web application stores password salted hashes ($P = H(pw|s)$) and that salt values (s) are also publicly known. If the salt is not known in advance, the browser can send an additional request to the server to obtain it.

4 Security Analysis

DVCert main's goal is to detect MITM attacks against SSL/TLS. DVCert achieves this by effectively binding the SSL/TLS layer to the application layer (i.e., channel binding [4, 39]). As a result, a MITM adversary trying to avoid detection by modifying the DCL is not only forced to compromise a CA to obtain a forged certificate but also to compromise each of the targeted domains to obtain users' authentication credentials.

An adversary can try to capture DVCert messages and use offline attacks to obtain user authentication credentials. However, the attacker needs to execute a MITM attack first to access DVCert messages. Thus, such attempts will be detected by DVCert. Furthermore, PAK's formal proofs of security for standard [8] and short exponents [29] (i.e., 384 bits) provide strong guarantees that the adversary will not learn password information from DVCert messages. DVCert modifications to PAK do not affect these proofs. For example, PAK and DVCert transmit the same number of hash values (2) over the network. The main difference is that DVCert uses one message less and uses the DCL as part of the computation of h_1.

We used ProVerif [6], an automatic cryptographic protocol verifier, to formally characterize DVCert. *Using ProVerif, we successfully demonstrated that DVCert does not leak password information (i.e., resilience to offline attacks).* Due to space limitations, ProVerif configuration details and results are available in DVCert's web site.

DVCert information stored in the browser or the server cannot be used to impersonate the user because DVCert does not provide user authentication. Therefore, DVCert offers resilience to server compromise similar to augmented PAKE protocols. The adversary can still use offline dictionary attacks against the stolen credentials, but the use of strong passwords can mitigate this risk.

The DCL includes fingerprints of certificates from third-party domains because these certificates cannot be validated directly (users do not share secrets with these domains). This is important because a MITM attack against a third-party SSL/TLS connection could be used to compromise the session with the web application (e.g., code injection attacks). The web application is responsible for maintaining the latest certificate information from third-party domains in the DCL. For example, the web application could rely on existing secure connections with third-party domains to obtain their certificate information. Alternatively, the application could rely on third-party validation mechanisms.

A concern with PAKE protocols is the risk of denial of service attacks due to the cost of public key operations. DVCert mitigates this risk by optimizing such operations without reducing security. For example, DVCert can use shorter exponents for better performance without affecting formal proofs of security. PAK allows the use of exponents with a minimum size of 384 bits (1024 bits DH group) [29] while maintaining a similar level of security. Another suggested optimization is the use of static parameters in the server (i.e., b, g^b and m_2) to reduce the number of operations (see Section 5). This technique affects the protocol's perfect forward secrecy property; however, DVCert does not require it (i.e., the session secret is not used for encryption). Finally, the web application could also monitor and limit the number of DVCert requests a user can make per day according to a domain policy.

5 Experimental Analysis

We implemented DVCert browser and server components (see Figure 2) to evaluate their performance and deployability. The DVCert browser component was implemented as an extension for Firefox 10.0.x and Firefox for mobile (Fennec) 4.03b. The extensions were written mainly in Javascript, but we also used C code for modular exponentiation operations through Firefox's js-ctypes API and the GMP library[2]. Approximately 500 lines of code were required for both extensions. The DVCert server component was implemented in PHP and required approximately 400 lines of code. More importantly, the DVCert server component is completely independent of the web application code; only access to the user database is required. PAK implementation details as well as test vectors were obtained from the RFC 5683 [9] and the ITU-T Recommendation X.1035 [21]. The experiments used a laptop (Apple MacBook Pro with dual core 2.53 GHz processor, 4GB of memory and Mac OS X 10.6) and a smartphone (Samsung Galaxy S 4G with a 1 GHz Cortex-A8 processor, 512 MB of memory and Android 2.2.1) as our clients. On the server side, we used a Ubuntu 10.10 server with 2 quad-core 2.00 GHz processors, 16 GB of memory and Gigabit Ethernet. The server was configured with Apache 2.2, PHP 5.3 and a 2048 bits RSA certificate. Finally, our prototype DVCert implementation is currently available for evaluation at `http://www.cc.gatech.edu/~idacosta/dvcert/index.html`.

Certificate validation operations using the DCL are inexpensive. For example, for each SSL/TLS connection, the browser executes one hash operation and one search operation. Assuming an ordered DCL, binary search is used to determine if a certificate is in the DCL with time $O(\log n)$, where the DCL's size n is in the order of tens of certificates. In addition, the size of the DCL is small (e.g., a SHA-1 certificate fingerprint requires only 160 bits). Hence, the impact on network bandwitdh due to the DCL is negligible. Therefore, our experimental

[2] Javascript-only DVCert add-ons for Firefox required an execution time at least one order of magnitude higher than add-ons using C native code for modular exponentiation, particularly in the smartphone. Ultimately, we envision DVCert to be implemented directly in the browser and using native code for its operations.

evaluation focused on the costs associated with DVCert transactions where more complex operations take place.

First, we measured the time required to generate a DVCert request (t_g) and the time required to verify the corresponding response (t_v) in the browser for different exponent sizes: 2048, 1024 and 384 bits. Morevoer, we used a DCL with one certificate fingerprint in all the experiments. Table 1 shows the results for 100 DVCert transactions per configuration using a laptop and a smartphone, including 95% confidence intervals. The results show that for 2048 bits exponents, an often recommended size for standard key exchange protocols [7], the browser required 26.78 ms and 440.58 ms of total computation time ($t_g + t_v$) on the laptop and on the smartphone respectively. While these computation times should not affect the user experience due to the low frequency of DVCert transactions, we can see that using 384 bits exponents decreased these times to 12.03 ms on the laptop (55.07% improvement) and 97.70 ms on the smartphone (77.82% improvement); thus, such delays are unlikely to be noticed by users.

Second, we measured the server response time using network traces for single HTTPS requests (i.e., our baseline) and HTTPS requests with DVCert. Each request retrieved a small HTML page ($\approx$ 500 bytes. We chose this small size to measured only the overhead added by SSL/TLS and DVCert). Moreover, our measurements did not include SSL/TLS setup times. For HTTPS request with DVCert, we evaluated different exponent sizes (2048, 1024 and 384 bits) and the use of dynamic (t_r) and static (t_{rsp}) server parameters. Based on these measurements, we estimated how much time the server spent on DVCert operations (t_d and t_{dsp}) by subtracting the baseline time from the HTTPS+DVCert server response times. The results for 100 DVCert transactions per configuration are shown in Table 2, including 95% confidence intervals. The most robust configuration, 2048 bits and dynamic parameters, required 10.71 ms of additional server computation time, while the most efficient configuration, 384 bits and static parameters, required around 0.54 ms (94.96% improvement). Thus, the most efficient DVCert configuration requires less time than serving a HTTPS request (1.17 ms) and it is smaller than the average network jitter in the US (0.67 ms [5]). Also, Table 2 shows how static parameters can reduce DVCert processing time on the server by at least 38%. Overall, these results show that DVCert operations have similar processing requirements to other server operations (e.g., SSL/TLS setup, HTTPS requests processing) while still maintaining robust security guarantees. Thus, DVCert should not degrade performance or increase the risk of DoS attacks.

Finally, we evaluated the overall impact of DVCert on server throughput in the hypothetical scenario where each SSL/TLS connection includes a DVCert transaction (i.e., upper bound). For this purpose, we measured the rate of HTTPS requests (using one SSL/TLS connection per request) and the rate of HTTPS+DVCert requests that the server can handle. As before, we evaluated DVCert with different exponent sizes (2048, 1024 and 384 bits) and one setup with static parameters and 384 bits exponents. The test load was generated with *httperf*, a HTTP traffic generator tool. Figure 4 shows the results of this

Table 1. DVCert request generation time (t_g) and response verification time (t_v), including 95% confidence intervals, on a laptop and on a smartphone for different exponent sizes.

Exp. Size	**Laptop t_g (ms)**	**Laptop t_v (ms)**	**Phone t_g (ms)**	**Phone t_v (ms)**
2048 bits	10.36 (±0.09)	16.42 (±0.29)	171.92 (±1.79)	268.66 (±9.64)
1024 bits	3.95 (±0.07)	9.55 (±0.14)	48.68 (±2.11)	71.88 (±7.87)
384 bits	3.26 (±0.09)	8.77 (±0.14)	33.58 (±0.72)	64.12 (±7.44)

Table 2. Server response time (t_r) for a HTTPS request and a HTTPS request with DVCert using dynamic and static parameters (t_{rsp}) and different exponent sizes. By subtracting the time of a single HTTPS request, we estimated the cost of DVCert operations with dynamic (t_d) and static (t_{dsp}) parameters and determined the percentage of improvement (% Imp.) due to static parameters.

Request Type	**t_r (ms)**	**t_d (ms)**	**t_{rsp} (ms)**	**t_{dsp} (ms)**	**% Imp. (t_{dsp})**
HTTPS only	1.17 (±0.01)	–	1.17 (±0.01)	–	–
DVCert 2048 bits	11.88 (±0.01)	10.71	6.66 (±0.01)	5.49	48.74%
DVCert 1024 bits	3.02 (±0.01)	1.85	2.20 (±0.01)	1.03	44.32%
DVCert 384 bits	2.04 (±0.01)	0.87	1.71 (±0.01)	0.54	37.93%

experiment for 10 measurements per point (300 in total), including 95% confident intervals. This figure shows that, even if every SSL/TLS connection uses a DVCert transaction, using 384 bits exponents allows a maximum throughput close to the one obtained using single HTTPS requests. Moreover, 1024 bit exponents could also allow a similar performance if static parameters are used (based on the results shown in Table 2). Thus, using 1024 bits exponents or shorter and static parameters reduces the risk of DoS attacks, eliminating the need for additional DoS defenses (e.g., client puzzles).

6 Discussion

6.1 DVCert Benefits

In addition to meeting the design goals described in Section 3.2, DVCert solves most of the problems hindering the deployment of third-party defenses against MITM attacks (see Section 2.3). First, *DVCert is easier to deploy and maintain.* In most scenarios, DVCert should not require additional infrastructure due to its low processing costs. Only minor modifications are required to add DVCert support to the web application and the browser (see Figure 2). For example, DVCert only needs access to the application's user database and certificate information (i.e., the DCL). Hence, DVCert can be deployed as an independent service without modifying any existing functionality in the application. In the browser, DVCert can also be implemented as an independent component that

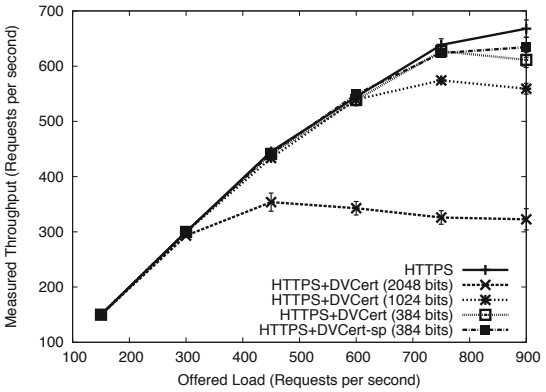

Fig. 4. Comparison of the web server throughput for single HTTPS request and HTTPS requests with DVCert in the hypothetical case that DVCert transactions are executed per SSL/TLS connection (i.e., upper bound). HTTPS+DVCert configurations used different exponent sizes and one configuration used static parameters (HTTPS+DVCert-sp).

only requires the certificate information used on each SSL/TLS connection and secure storage for the password salted hashes and DCL data. Moreover, by relying on passwords, users do not need to deal with additional secrets or devices and can benefit from DVCert on a wider range of platforms. Second, *DVCert has a simpler trust model.* It relies on existing trust relationships between users and web applications; hence, users do not need to assess and establish new trust relationships with third-parties. Third, *DVCert does not introduce new privacy risks.* User browsing activity is not revealed to third-parties when a certificate is validated using DVCert. This property is particularly important for users with high privacy and anonymity requirements (e.g., Tor users). Fourth, *certificate revocation is simpler.* For instance, a certificate can be revoked by just removing it from the DCL. Thus, there is no need for mechanisms such as CRLs and OCSP, both criticised due to their ineffectiveness [24]. Fifth, *DVCert is more resilient to compromise than third-party approaches.* Third-party solutions can vouch for certificates belonging to a large number of domains. However, if compromised, then all the protected domains could be affected by MITM attacks. In contrast, DVCert is deployed independently per domain; thus, attacks against one domain will not affect other domains. Finally, *DVCert is compatible with captive portals in certain scenarios.* For instance, DVCert could verify the certificates of captive portals that already share a secret with the user (e.g., Wi-Fi provider account) or where the user receives a shared secret via a secondary channel (e.g., a receipt).

6.2 DVCert Limitations

DVCert allows web applications to vouch for their certificates using existing authentication credentials. Thus, DVCert can only protect web applications where

the user has an account and a shared secret. However, this is not a major limitation because most of the web applications that are likely to be targeted by adversaries (e.g., sites with private information) require authentication credentials. A related case are web applications that rely on federated identity management (e.g., OpenID) or Single sign-on (SSO) systems. Here, users share a password with an identity provider instead of the web application. Still, DVCert can be extended to validate certificates in such scenarios. For instance, the web application can provide its DCL to the identity provider during the login process. Then, the browser can execute a DVCert transaction to obtain not only the DCL of the identity provider but also of the targeted application. We plan to explore this idea in our future work. Another limitation is that DVCert cannot be used to protect the first connection to a web application. DVCert is by design a trust-on-first-use (TOFU) [38] mechanism such as the SSH protocol. Therefore, when registering to a web application for the first time, users can only rely on CA signatures and other third-party mechanisms to validate certificates. However, for most scenarios, it is unlikely that adversaries will be monitoring users before they have created an account with a web application. Moreover, applications with high security requirements could also use secondary channels to protect the user registration process.

7 Related Work

Multiple browser-based mechanisms have been proposed to detect forged certificates. For instance, browser extensions can keep track of the certificates used by the browser and can detect certificate changes [1, 36]. While simple, the effectiveness of this approach is affected by false positives and lack of user training. A related technique, known as certificate pinning [16], uses a white-list of certificates for important domains that are hardcoded in the browser. This solution is less prone to false positives; however, it is neither flexible nor scalable. A more robust approach is the use of secondary channels such as cellular networks [33] and Tor [3] to obtain additional copies of the server certificate. Unfortunately, this approach is difficult to deploy and can introduce significant delays.

Most research in the area of MITM defenses focuses on using additional third-parties to improve or replace the CA trust model. For example, mechanisms such as Perspectives [38] and Convergence [30] allow users to choose multiple network notaries that can complement or replace CAs signatures. The Mutually Endorsing CA Infrastructure (MECAI) [14] proposes a similar approach, but instead of introducing new notaries, MECAI uses existing CAs as notaries. A different technique is presented by the Electronic Frontier Foundation (EFF) Sovereign Keys (SK) project [12]. In SK, domain certificates include an additional integrity signature created with the domain's sovereign key. To verify this signature, browsers can obtain the corresponding sovereign key from a semi-centralized, append-only public data structure. Google's Certificate Transparency (CT) [25] proposal also relies on a similar data structure, but instead of storing keys, it stores records of each certificate emitted by a CA; thus, browsers can check this public audit log

to validate they are using the correct certificate. The IETF DNS-based Authentication of Named Entities (DANE) working group [20] is developing protocols that use secure DNS (DNSSEC) extensions to bind certificates to domain names. Finally, while third-party based solutions offer several benefits, their adoption has been hindered by multiple problems such as deployment and operational costs, lack of user training, false positives and others (see Section 2.3).

To a lesser degree, researchers have also explored the use of shared secrets (e.g., passwords) to defend against MITM attacks. For example, the TLS-SRP protocol [37] uses SRP [40] for mutual authentication and SSL/TLS key derivation based on the user's password (i.e., certificates and CAs are not required). However, TLS-SRP requires inter-layer communication between the application and the SSL/TLS stack, breaking SSL/TLS transparency. A different technique is to use shared secrets for channel binding [39], as proposed in the Session Aware (TLS-SA) user authentication protocol [32]. To detect MITM attacks, TLS-SA uses authentication codes based on user credentials and SSL/TLS session information, effectively binding the application and SSL/TLS layers. TLS-SA, however, requires client certificates and hardware tokens to resist offline dictionary attacks, affecting its adoption. Finally, the Mutual Authentication Protocol for HTTP [31] also combines user authentication with SSL/TLS channel binding, but it relies on the user's password instead of client certificates. To provide mutual authentication and prevent offline guessing attacks, this mechanism relies on the direct implementation of a PAKE protocol. However, this mechanism requires additional server state, only protects the login connection and requires changes to the browser and web application login UI (a significant challenge for deploying PAKE-based protocols [15]).

8 Conclusions

As recent incidents have demonstrated, adversaries are exploiting weaknesses in the CA trust model to compromise communications protected by SSL/TLS via MITM attacks. This trend is likely to accelerate as more and more web applications adopt SSL/TLS to protect all their communications. Currently proposed solutions face multiple challenges due to their complexity and deployment and operational costs; thus, they are unlikely to be widely available in the near future. We present DVCert, a practical mechanism that relies on previously established shared secrets to allow the web application to directly and securely vouch for the authenticity of its certificates. By using a single round-trip transaction with the web application, based on a modified PAK protocol, the browser learns the information required to locally verify all the certificates that could be used during a session with the application. Our experimental analysis shows that DVCert transactions require little execution time on the server and the browser; therefore, they should not have a serious impact on server performance or user experience. Finally, DVCert could be extended to protect not only the integrity of SSL/TLS certificates but also other application's resources such as Javascript code and binary objects. We intend to explore this approach in our future work.

Acknowledgments. This work was supported in part by the US National Science Foundation (CAREER CNS-0952959). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would also like to thank William Enck for his helpful comments.

References

1. Certificate Patrol (2010), http://patrol.psyced.org/
2. Adams, C., Farrell, S.: RFC 2510 - Internet X.509 Public Key Infrastructure Certificate Management Protocols (1999), https://tools.ietf.org/html/rfc2510
3. Alicherry, M., Keromytis, A.D.: DoubleCheck: Multi-path Verification Against Man-in-the-Middle Attacks. In: Proceedings of the IEEE Symposium on Computers and Communications (2009)
4. Altman, J., Williams, N., Zhu, L.: RFC 5929 - Channel Bindings for TLS (2010), http://tools.ietf.org/html/rfc5929
5. AT&T: Network Averages (2012), http://ipnetwork.bgtmo.ip.att.net/pws/averages.html
6. Blanchet, B.: ProVerif: Cryptographic Protocol Verifier in the Formal Model, http://www.proverif.ens.fr/
7. BlueKrypt: Cryptographic Key Length Recommendation (2012), http://www.keylength.com/
8. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
9. Brusilovsky, A., Faynberg, I., Zeltsan, Z., Patel, S.: RFC 5683 - Password-Authenticated Key (PAK) Diffie-Hellman Exchange (2010), http://tools.ietf.org/html/rfc5683
10. Dierks, T., Rescorla, E.: RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2 (2008), http://tools.ietf.org/html/rfc5246
11. Eckersley, P., Burns, J.: The (Decentralized) SSL Observatory. In: USENIX Security Symposium (2011) (Invited Talk)
12. Electronic Frontier Foundation (EFF): The Sovereign Keys Project (2011), https://www.eff.org/sovereign-keys
13. Ellison, C., Schneier, B.: Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. Computer Security Journal 16(1), 1–7 (2000)
14. Engert, K.: MECAI (2011), http://kuix.de/mecai/
15. Engler, J., Karlof, C., Shi, E., Song, D.: Is It Too Late for PAKE? In: Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (2009)
16. Evans, C., Palmer, C.: Certificate Pinning Extension for HSTS (2011), http://www.ietf.org/mail-archive/web/websec/current/pdfnSTRd9kYcY.pdf
17. Freier, A., Karlton, P., Kocher, P.: RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0 (2011), https://tools.ietf.org/html/rfc6101
18. Goodin, D.: Web Authentication Authority Suffers Security Breach (2011), http://www.theregister.co.uk/2011/06/21/startssl_security_breach/
19. Gutman, P.: PKI: It's Not Dead, Just Resting. Computer 35(8), 41–49 (2002)
20. Hoffman, P., Schlyter, J.: IETF Internet-Draft: Using Secure DNS to Associate Certificates with Domain Names For TLS (draft-ietf-dane-protocol-06) (2011), http://tools.ietf.org/html/draft-ietf-dane-protocol-06

21. International Telecommunication Union: ITU-T Recommendation X.1035: Password-Authenticated Key Exchange (PAK) Protocol (2007), http://www.itu.int/rec/T-REC-X.1035/en
22. Keizer, G.: Hackers May Have Stolen Over 200 SSL Certificates (2011), https://www.computerworld.com/s/article/9219663/Hackers_may_have_stolen_over_200_SSL_certificates
23. Kirk, J.: KPN Stops Issuing SSL Certificates After Possible Breach (2011), https://www.pcworld.com/businesscenter/article/243275/kpn_stops_issuing_ssl_certificates_after_possible_breach.html
24. Langley, A.: Revocation Doesn't Work (2011), http://www.imperialviolet.org/2011/03/18/revocation.html
25. Laurie, B., Langley, A.: Certificate Authority Transparency and Auditability (2011), http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf
26. Leyden, J.: Inside 'Operation Black Tulip': DigiNotar Hack Analysed (2011), http://www.theregister.co.uk/2011/09/06/diginotar_audit_damning_fail/
27. Leyden, J.: Trustwave Admits Crafting SSL Snooping Certificate (2012), http://www.theregister.co.uk/2012/02/09/tustwave_disavows_mitm_digital_cert/
28. MacKenzie, P.: The PAK suite: Protocols for Password-Authenticated Key Exchange. In: IEEE P1363.2: Password-Based Public-Key Cryptography (2002)
29. MacKenzie, P.D., Patel, S.: Hard Bits of the Discrete Log with Applications to Password Authentication. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 209–226. Springer, Heidelberg (2005)
30. Marlinspike, M.: Convergence (2011), http://convergence.io/
31. Oiwa, Y., Takagi, H., Watanabe, H., Suzuki, H.: PAKE-based Mutual HTTP Authentication for Preventing Phishing Attacks (Poster). In: Proceedings of the International Conference on World Wide Web, WWW (2009)
32. Oppliger, R., Hauser, R., Basin, D.: SSL/TLS Session-Aware User Authentication. Computer 41(3), 59–65 (2008)
33. Parno, B., Kuo, C., Perrig, A.: Phoolproof Phishing Prevention. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006. LNCS, vol. 4107, pp. 1–19. Springer, Heidelberg (2006)
34. Richmond, R.: An Attack Sheds Light on Internet Security Holes (2011), http://www.nytimes.com/2011/04/07/technology/07hack.html
35. Singel, R.: Law Enforcement Appliance Subverts SSL (2010), http://www.wired.com/threatlevel/2010/03/packet-forensics/
36. Soghoian, C., Stamm, S.: Certified Lies: Detecting and Defeating Government Interception Attacks against SSL (Short Paper). In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 250–259. Springer, Heidelberg (2012)
37. Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: RFC 5054 - Using the Secure Remote Password (SRP) Protocol for TLS Authentication (2007), http://tools.ietf.org/html/rfc5054
38. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: Improving SSH-style Host Authentication with Multi-path Probing. In: Proceedings of the USENIX Annual Technical Conference, ATC (2008)
39. Williams, N.: RFC 5056 - On the Use of Channel Bindings to Secure Channels (2007), http://tools.ietf.org/html/rfc5056
40. Wu, T.: The Secure Remote Password Protocol. In: Proceedings of the Network and Distributed System Security Symposium (1998)