VeriKey: A Dynamic Certificate Verification System for Public Key Exchanges

Brett Stone-Gross, David Sigal, Rob Cohn,
John Morse, Kevin Almeroth, and Christopher Kruegel

Department of Computer Science,
University of California, Santa Barbara,
{bstone, dsigal, rcohn, morse, almeroth, chris}@cs.ucsb.edu

Abstract. This paper presents a novel framework to substantiate self-signed certificates in the absence of a trusted certificate authority. In particular, we aim to address the problem of web-based SSL man-in-the-middle attacks. This problem originates from the fact that public keys are distributed through insecure channels prior to encryption. Therefore, a man-in-the-middle attacker may substitute an arbitrary public key during the exchange process and compromise communication between a client and server. Typically, web clients (browsers) recognize this potential security breach and display warning prompts, but often to no avail as users simply accept the certificate since they lack the understanding of Public Key Infrastructures (PKIs) and the meaning of these warnings. In order to enhance the security of public key exchanges, we have devised an automated system to leverage one or more vantage points of a certificate from hosts that have distinct pathways to a remote server. That is, we have a set of distributed servers simultaneously retrieve the server's public key. By comparing the keys received by peers, we can identify any deviations and verify that an attacker has not compromised the link between a client and server. This is attributable to the fact that an attacker would have to compromise all paths between these vantage points and the server. Therefore, our technique greatly reduces the likelihood of a successful attack, and removes the necessity for human interaction.

1 Introduction

As e-commerce and subsequent online transactions emerged on the Internet, there became a need to protect sensitive communication. This requirement was partially fulfilled by introducing public key cryptography to secure key exchanges. Public key cryptography utilizes certificates to bind an entity to a specific public key with a digital signature. This digital signature may belong to a well-known Certificate Authority (CA) or the creator of the certificate. The former is generally regarded as more secure, but has been exploited previously due to improper implementations [3]. The latter poses a greater security risk, because there is no way (other than manually verifying the certificate's fingerprint) to confirm the identity of the owner. As a result, a malicious host can exploit this uncertainty through the use of a man-in-the-middle attack by intercepting and altering a certificate to impersonate the same party with another key known to the attacker. When the public key on the certificate is replaced with the key of an attacker, the integrity of the encrypted session is compromised if the forged certificate is accepted.

The most popular public key encryption protocol for the World Wide Web has been the Secure Sockets Layer (SSL) and its successor, the Transport Layer Security (TLS) protocol. While these protocols have proven to be relatively effective, they are also vulnerable to man-in-the-middle attacks in situations where the user or the client implementation is not able to detect the fraudulent certificate. These attacks are normally a consequence of other insecure protocols that allow a malicious host to easily become a man-in-the-middle. Some of these exploitable protocols include the Address Resolution Protocol (ARP), Domain Name System (DNS), and Dynamic Host Configuration Protocol (DHCP). Wireless networks are also particularly vulnerable since attackers can easily deploy rogue access points, modify unencrypted link layer frames, and are susceptible to most switched Local Area Network (LAN) attacks. Only in large networks with sophisticated Intrusion Detection Systems (IDS) are these attacks routinely detected and prevented.

In order to prevent SSL man-in-the-middle attacks on the Internet, several companies operate as certificate authorities to digitally sign X.509 certificates. Certificates from these CAs are assumed to be secure, since their public keys are well-known and are included with standard web browser distributions. Therefore, a web browser only needs to confirm that the digital signature on a certificate matches one of these trusted CAs. For the majority of cases, the certificates signed by these CAs function well. However, the service that these CAs provide comes at a fairly high cost, which deters some web server administrators from purchasing certificates from them. Current rates for a single domain certificate cost approximately \$100-\$200 USD per year, and a wildcard domain certificate costs about \$500-\$1,000 USD per year[1]. To avoid this expensive surcharge, smaller web sites often create their own self-signed certificates, or purchase a certificate for only a single domain.

When a certificate is self-signed or if the common name and fully qualified domain name do not correspond, nearly all web browsers will display a prompt that requests user input on whether to accept the certificate. These warning dialogs occur even when visiting popular sites like `https://amazon.com` and `https://bankofamerica.com` because the common name on their certificates are registered only to `https://www.amazon.com` and `https://www.bankofamerica.com`, respectively. The problem derives from the fact that many web users do not understand the meaning of these "cryptic" messages, and will accept almost any certificate [21]. Therefore we believe that it is important to develop a system that does not require user attention for normal users, while offering supplemental knowledge for expert computer users to determine the legitimacy of a certificate, without having prior knowledge of the remote web server's public key.

[1] http://www.digicert.com/

In this paper, we propose a novel solution to augment the security of SSL certificate exchanges. The primary objective of our system is to remove certificate prompts from the web browser, and instead rely on what peers elsewhere on the Internet observe. We term these outside peers *verification servers* and refer to them as such throughout the paper. By combining multiple views from verification servers, our system greatly reduces the likelihood that an attacker can intercept and inject their own certificate during the SSL handshake without being detected. This follows from our method to select verifications servers such that they have *different* pathways to a remote web server. As a result, our approach considerably minimizes the potential man-in-the-middle attack vectors because an attacker would have to compromise all these paths to launch a successful man-in-the-middle attack. To select servers that have different paths, our system leverages Autonomous System (AS) level topological mappings. Another benefit our of design is that it is based on existing protocols. Therefore, our system can be readily deployed, since it requires no modifications to web servers, and can be implemented in current web browsers through an extension or plug-in. In order to evaluate our system's relative effectiveness and performance, we deployed our prototype on PlanetLab [15]. Further applications of our design also extend to other non-web-based public key protocols including Secure Shell (SSH), Internet Message Access Protocol (IMAPS), and Secure Copy (SCP).

The remainder of this paper is organized as follows. In Section 2, we introduce relevant work that has been previously performed. Section 3 explains how and why SSL man-in-the-middle attacks work and the potential attack vectors. Section 4 presents our design considerations, system architecture, and certificate verification process. We then provide an evaluation of our system and potential extensions in Section 5. Finally, Section 6 concludes with a summary of our contributions.

2 Related Work

There are various protocols from the link layer to the application layer that have been exploited using man-in-the-middle attacks. As a result, there have been numerous studies to detect and prevent the root causes of each vulnerability. One of the most prominent man-in-the-middle attacks on a LAN is ARP poisoning [20]. This link layer protocol is insecure since it neither provides any form of message authentication nor maintains any state information. In order to mend these vulnerabilities, several solutions have been proposed that authenticate hosts and record the bindings of link layer MAC addresses to network layer IP addresses [2][10][12]. The problem with these solutions is that they are difficult to deploy due to added infrastructure, operating system modifications for all connected hosts, and complexities in key distribution for authentication.

Another common local man-in-the-middle attack exploits the DHCP protocol, which is used to automate network configurations. Since DHCP lacks message authentication, an attacker can impersonate and forge DHCP replies to other hosts, thus manipulating the victim's IP address, gateway address, and

DNS server information. To mitigate this vulnerability, several authentication and access control systems have been proposed [1][6][11]. These approaches share the same limitations as the ARP prevention systems in that all hosts require operating system modifications to access the network. Moreover, they require the configuration and setup of authentication servers.

Potentially the most dangerous man-in-the-middle vulnerability stems from the weaknesses of DNS. Due to the hierarchical structure of DNS, an attacker may spoof DNS responses that may affect not only a local network, but also remote networks. To address these shortcomings, the Domain Name System Security Extensions (DNSSEC) protocol was proposed [8]. DNSSEC is designed to prevent manipulation of DNS queries and replies via authentication and data integrity. Current deployment of this protocol has been impeded by a lack of backward-compatibility with existing DNS servers and hosts. ConfiDNS is another system that monitors DNS replies from multiple vantage points, with the goal of identifying inconsistencies [16]. However, ConfiDNS is only effective against attacks on local DNS servers, and has difficulty with discrepancies produced by DNS load balancing.

An alternative approach to improve current PKIs was introduced by Zhou *et al.* through a distributed online certificate authority with a fault-tolerant algorithm that uses Byzantine quorums, called COCA [22]. While this solution is elaborate, there are numerous complexities in the distributed infrastructure that make it impractical to deploy, configure, and maintain.

Unfortunately, all of the preceding solutions require extensive modifications to hosts, lack support for non-compliant hosts, or both. Instead of developing multiple protocols to address all possible man-in-the-middle attack vectors, we take a different approach and provide a method to avert these attacks at the application level. This solution permits users of our system to install a web browser plug-in, which will run silently in the background. This technique makes our system backward-compatible with all existing protocols. When we combine this method with lightweight verification servers, we are able to attain a considerably more deployable solution.

3 SSL Man-in-the-Middle Attack Overview

A man-in-the-middle attack occurs when a malicious host deceives others into forwarding their traffic to them by impersonating the intended receiver. The potential for these attacks exist in virtually all networks that use unauthenticated communication (*e.g.*, no encryption and/or message integrity checks). Consequently, these packet manipulations facilitate man-in-the-middle attacks on encrypted protocols that exchange public keys through certificates. This weakness is due to the fact that most implementations of cryptographic algorithms allow user interaction to accept self-signed certificates and their associated public keys. Tools that automate these exploits include `webmitm`[7] and `ettercap`[9]. An attacker using these tools is able to redirect all of a victim's traffic to himself instead of the original destination. Figure 1 demonstrates a classic SSL man-in-

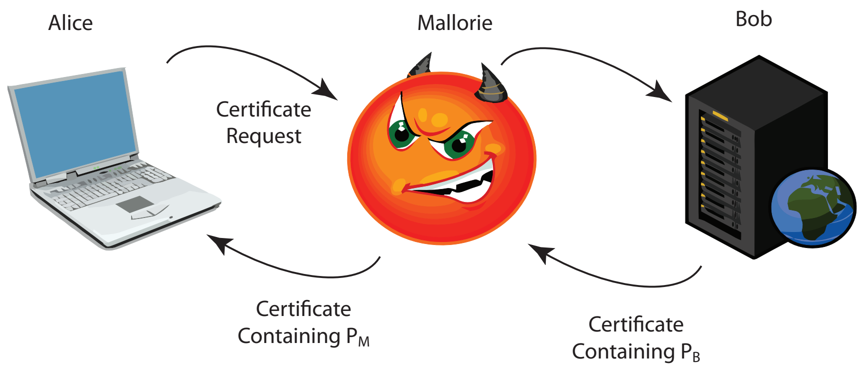


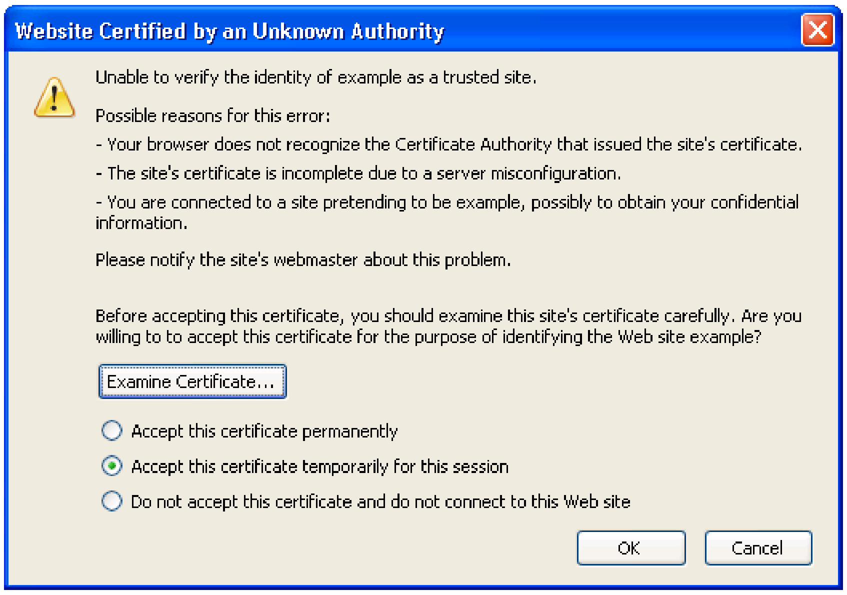
Fig. 1. Example of an SSL Man-in-the-Middle Attack.

the-middle attack. In this example, Alice and Bob want to communicate with each other using SSL. The first step in the SSL handshake requires Alice to contact Bob and request his certificate containing his public key, P_B. However, without Alice's knowledge, Mallorie, who is on the same physical network as Alice, has previously poisoned her ARP cache, causing Alice to address all of her packets to Mallorie. When Mallorie observes Alice's SSL request, she is able to intercept her request and make her own SSL connection to Bob. Mallorie then replies to Alice's request with her own public key, P_M. Alice is prompted by her web browser that the certificate that she received was valid, but not signed by a trusted CA. Unfortunately, Alice does not know the meaning of the warning and accepts the forged certificate. Alice now begins encrypting her traffic to Bob using the public key of Mallorie, P_M. Consequently, Mallorie is able to decrypt, monitor, and modify all communications before relaying messages between Alice and Bob.

As illustrated in the preceding example, most current web browsers display warning prompts when certificates cannot be validated. These dialogs commonly appear in the following cases:

– A certificate has expired.
– A certificate is signed by a trusted CA and belongs to a domain (without wildcards), but is not associated with any subdomain (*e.g.*, https://bankofamerica.com *vs.* https://www.bankofamerica.com).
– The common name on the certificate does not match the domain name of the host.
– A certificate is not signed by a trusted certificate authority (*e.g.*, a self-signed certificate or a certificate signed by a non-trusted CA).

Figure 2 shows an example of a typical certificate mismatch dialog displayed by web browsers.


Website Certified by an Unknown Authority

Unable to verify the identity of example as a trusted site.

Possible reasons for this error:

- Your browser does not recognize the Certificate Authority that issued the site's certificate.
- The site's certificate is incomplete due to a server misconfiguration.
- You are connected to a site pretending to be example, possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site example?

Examine Certificate...

( ) Accept this certificate permanently
[✓ Accept this certificate temporarily for this session]
( ) Do not accept this certificate and do not connect to this Web site

OK Cancel

Fig. 2. Mozilla Firefox warning that a certificate is not signed by a trusted CA.

4 System Architecture

In this section, we first present the considerations that went into our design, as well as the assumptions that were made. Next, we introduce our system's architecture and explain each component in detail. We then discuss our certificate verification process. Finally, we describe how clients determine which verification servers to contact.

4.1 Design Considerations

Before we examine the details of our design, we first present our security threat model and our assumptions.

The basis for our design is derived from our security threat model shown, in Figure 3. We believe that because most users of web applications are not experts in public key cryptography, they are the weakest link in the process when given the power to make a decision to accept a certificate. In contrast, verification and web servers are generally deployed by security-conscious network administrators (*i.e.*, experts), who are knowledgeable about certificates, PKIs, and common network vulnerabilities. Since attackers are more likely to exploit the easiest target, which is frequently the user's inexperience with certificates, it is critical to assist the client in making the correct decision to reduce this vulnerability. Therefore, our system focuses particularly on protecting the client from these man-in-the-middle attacks during public key exchanges.

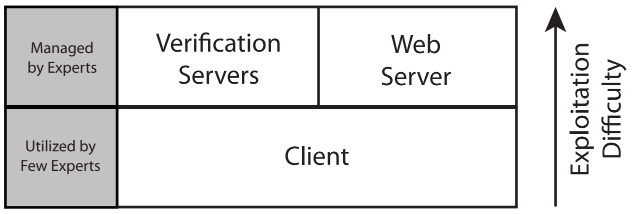


Fig. 3. VeriKey Security Threat Model.

In our design, we have made the following assumptions.

1. Web servers and verification servers are on relatively secure networks (*i.e.*, networks that perform some form of monitoring) and have not all been compromised. Depending on the number of verification servers utilized, our system may still function properly if one or more verification servers have been compromised.
2. The client must be able to authenticate and communicate securely with the verification servers. This requirement can be put into practice by distributing the certificates of verification servers to the client as part of a web browser extension. This serves as a pre-shared key mechanism so that the client can authenticate the verification server as well as communicate over a secure, encrypted connection. Alternatively, verification servers may obtain certificates from well-known CAs and eliminate the need to acquire each verification server's public key from the browser plug-in.
3. There exists at least one non-compromised pathway to a web server. Our system depends on this notion to identify inconsistencies reported by different verification servers. Therefore, if a man-in-the-middle is located on or near the web server's network, or if only one pathway to the web server exists, our system would not be able to detect an attack. We are confident, however, that in most cases this assumption holds because an attacker will normally exploit vulnerable client networks (*e.g.*, an insecure wireless hotspot). We discuss the impacts of this assumption later in Section 5.4.

4.2 Certificate Verification Components

In this section, we present the components that make our system effective in subverting man-in-the-middle attacks based on the preceding set of assumptions. As previously mentioned, the general idea of our system is to add an extra layer of security to public key exchanges by coalescing diverse views from trusted peers, and verifying that they are the same. We achieve this verification with an automated process, which is critical in removing user interaction that may otherwise compromise security. The fundamental principles that we incorporated in our design are based on the inherent lack of user understanding about the operation of public key cryptography. Therefore, the client-side of the verification system

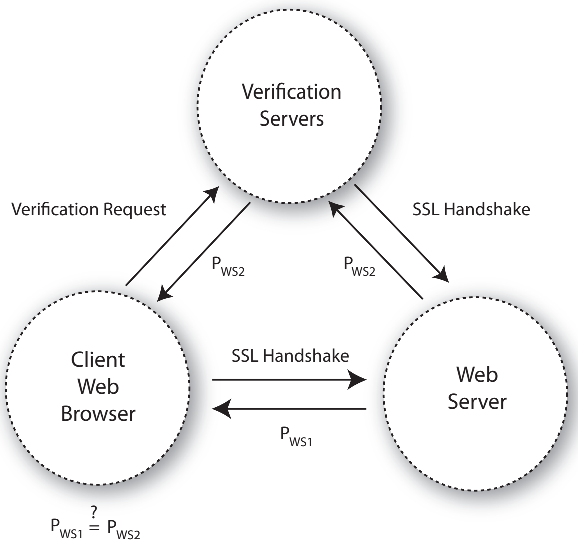


Fig. 4. Components of the VeriKey System Architecture.

must be extremely easy to use. On the server-side, simplicity and compatibility with existing web servers is essential. Hence, a web server should be able to interface with our system with no modifications. For certificate validation, we have developed a lightweight verification server that provides the client with its own perception of a remote certificate.

As shown in Figure 4, there are three main components of our system that implement our objectives: the client, verification servers, and web server. The client that we refer to includes standard web browsers with extension support (*e.g.*, a Mozilla Firefox/Opera plug-in or Internet Explorer add-on). The second component can be almost any existing web server with SSL support such as Apache, IIS, or Tomcat. These web servers will function with our system with no alterations or additional modules. The verification servers operate as the intermediary between the client and web server and handle certificate exchanges, caching, and verification.

4.3 System Deployment

Deploying our system is straightforward. The only requirement for a client is to install a web browser plug-in. We also provide the option for advanced users to deploy their own verification servers (although not necessary). The deployment process to interface with VeriKey is described below.

Client. The client must perform a one-time installation of our web browser extension. This extension contains a pairwise set of IP addresses and certificates containing the public keys for the default verification servers $\{\{\mathrm{IP}_1, \mathrm{P}_{V_1}\}, \{\mathrm{IP}_2, \mathrm{P}_{V_2}\}, \{\mathrm{IP}_n, \mathrm{P}_{V_n}\}\}$. As mentioned previously, verification servers may also attain a digitally signed certificate from a well-known CA such as VeriSign. In that case, only the distribution of their IP addresses is necessary. In addition, pre-computed AS topology maps are bundled that assist in verification server selection (discussed later in Section 4.5). The extension also has the ability to automatically update, revoke, and add new verification servers. Expert users may configure extra security requirements and have the option to integrate the information obtained during the verification process into the default security warning rather than allowing an automatic decision to be made on their behalf.

Verification Server. As previously discussed, more advanced users have the ability to deploy and configure their own verification servers. The web browser extension can then be configured to update its verification server set to point to these new custom servers. These verification servers may be deployed at research institutions, corporations, and other large organizations.

4.4 Certificate Integrity and Verification

In order to efficiently validate certificates, we have devised the following methodology as demonstrated in Figure 5.

Case 0: The initial step involves retrieving the web server's certificate. This exchange ensues during the SSL handshake, and the client must then confirm whether a trusted CA has signed the certificate.

Case 1: If the certificate has been signed by a trusted CA, the common name on the certificate is compared with the domain name of the web server.

– If the common name matches the domain name of the web server, the certificate should be trusted and the SSL connection can proceed normally. We ignore any mismatch between a domain and its possible subdomains, which commonly occurs when web sites do not purchase wildcard certificates and thus trigger warning messages. We justify this rationale based on the fact that it would be extremely difficult for an attacker to acquire a legitimate certificate for a subdomain that belonged to another entity.
– If the common name does not match the domain of the web server, the certificate (although valid), should not be trusted and the SSL connection between the client and web server should be blocked. This would occur when a man-in-the-middle has obtained a legitimate signed certificate from a trusted CA but for a different domain. Thus the man-in-the-middle could inject his own certificate (*e.g.*, www.hacker.com) during communications with another web server (*e.g.*, www.bank.com).

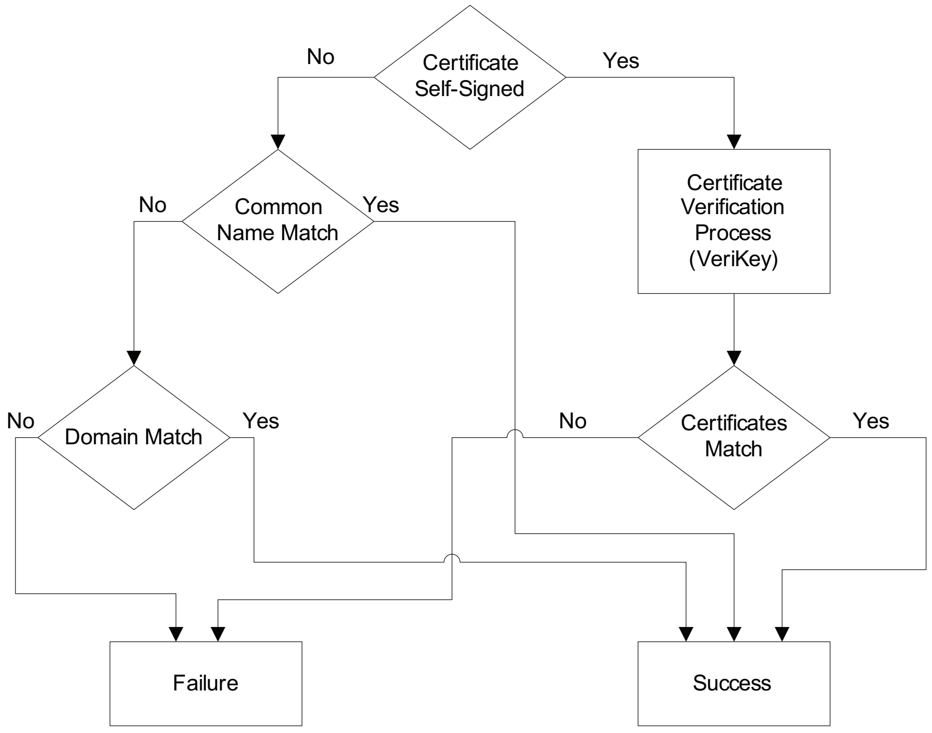


Fig. 5. Flow diagram depicting the certificate integrity process.

Case 2: If the certificate has not been signed by a trusted CA (*e.g.*, is self-signed), then the VeriKey certificate verification process will commence accordingly. Each of the following steps are illustrated in Figure 6.

1. **Client communication with verification servers.** The client connects to a number of verification servers. How these servers are selected will be discussed in the following section. After completing the handshake, the client verifies that the SSL handshake results in the reception of the correct public key of the verification server, P_v. If the certificate does not match, there is a man-in-the-middle between the client and verification server. Otherwise, the client sends the full domain name and IP address of the web server to the verification server. The purpose of sending the pair (domain, IP) is to enable the verification server to determine if its own DNS resolution matches that of the client. In addition, these pairs are required when a single domain name may resolve to multiple IP addresses.
2. **Certificate Request.** The verification server checks its internal cache to see if it has previously retrieved the certificate. If the public key is not found in the cache, it connects to the web server to retrieve the server's certificate.

Otherwise, the verification server forwards the cached public key of the web server to the client and Step 3 is omitted.

3. **Certificate Exchange.** The verification server (a) connects to the web server to retrieve its public key and (b) forwards it to the client.
4. **Public Key Verification.** At this stage, the client can now compare the certificate and public key of the server. If the public keys match, the client can communicate with the web server. Otherwise, there is a problem with the connection (*e.g.*, a man-in-the-middle) and the client will be notified that a potential security risk has been identified and the web server connection will be terminated. If the client's browser plug-in has been configured with higher security requirements, multiple verification servers will be utilized to verify the public key of the web server. When this option is chosen, a variable-based threshold τ is used in determining the number of public key matches μ that are required from the verification servers such that $\mu > \tau$. When the plug-in is configured in expert mode, the observed results are presented to the user, enabling them to make a manual decision instead of aborting the connection. In Section 5.5, we propose a mechanism to use the verification server as an SSL proxy, which may offer an alternative secure communication channel to the web server when a man-in-the-middle has been detected.
5. **(Optional) Verification Status Report.** This last step is optional, but allows a client to provide feedback to verification servers on the results of the certificate verification. This step enables the verification servers to determine if a certificate on the remote web server has changed and needs to be updated as well as to construct records of man-in-the-middle attacks.

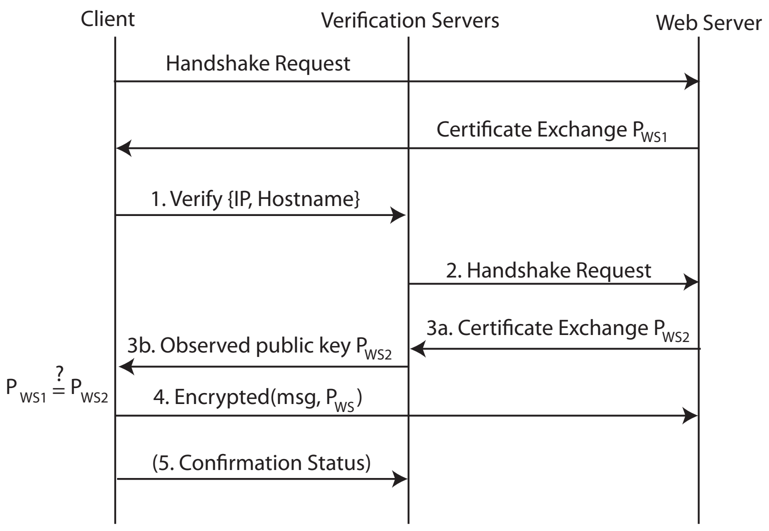

Client
Verification Servers
Web Server
Handshake Request
Certificate Exchange P_{WS1}
1. Verify {IP, Hostname}
2. Handshake Request
3a. Certificate Exchange P_{WS2}
3b. Observed public key P_{WS2}
$P_{WS1} \overset{?}{=} P_{WS2}$
4. Encrypted(msg, P_{WS})
(5. Confirmation Status)

Fig. 6. The VeriKey certificate verification process.

4.5 Verification Server Selection

The architecture of our system provides support for utilizing multiple verification servers. A client may elect to query several verification servers for their view of a particular web server's certificate, which in the general case, will enhance a client's perspective. However, this technique may neither be the most effective nor efficient method to validate certificates. There is the possibility that one or more verification servers will share the same pathway to the web server. This scenario would occur if an attacker was positioned on one of these shared pathways between the client and server. Thus, the security of the verification process does not always increase with the number of responses from additional verification servers. In order to determine the number and locations of verification servers to query, we have developed our own selection algorithm.

Before presenting our verification server selection technique, we first introduce other potential selection options and the relevant effects. In order to assist our analysis, we define the distance between two hosts as the number of Autonomous System (AS) links on the shortest path between them and denote this as $\mathbf{d}(\text{source, destination})$. The selection procedure may optimize for latency, limited resources, and/or security. The easiest way to reduce the latency of the verification process is to minimize the sum of the distances between client, verification server, and web server (*i.e.,* min $\{\mathbf{d}(C, V_{Si}) + \mathbf{d}(V_{Si}, W_S)\}, \forall : V_{Si}$). However, this method is naïve, because when the combined distance is minimal there is a higher probability that more than one of these hosts have paths that overlap.

If limiting the amount of resources (*e.g.*, CPU time and bandwidth) is an important consideration, a client can randomly select a single verification server. This approach has several limitations including an increased response time, particularly if the verification server is located far from the client and web server. Our approach, however, enhances security while reducing latency and resources when selecting a verification server. We achieve this functionality by computing the shortest AS pathways in advance and then utilize the information to compare the overlap among a set of verification servers. This data can be collected from publicly available Internet topology maps that analyze BGP routing dynamics to discover AS adjacencies. The data sources and the details of our implementation are discussed later in this section.

We denote the path between client and web server as $C \Rightarrow W_S$ and the path between verification server and web server as $V_S \Rightarrow W_S$, where C, V_S, and W_S represent the client, verification server(s), and web server respectively. We first calculate the shortest path between the AS of the source and the AS of the destination (given our topology information). After this computation, we select the verification server that has the *least amount of link overlap* between the two paths (*i.e.*, $min\{C \Rightarrow W_S \cap V_{Si} \Rightarrow W_S\}$, $\forall : V_{Si}$). If more than one shortest pathway exists between hosts, we calculate the average overlap between paths according to the equation $\frac{1}{n}\sum_{i=1}^{n} \lambda_i$ where λ is the number of links that overlap. This rationale follows from the fact that an attacker cannot influence or predict the exact path between any two hosts, and packets are more probable to follow shorter pathways. If there are multiple paths that have equivalent overlap, one path will be selected randomly from the potential paths with a uniform probability.

An example of our verification selection algorithm is shown in Figure 7. In this instance, the client is located within the UCSB AS, and is connecting to a Bank of America web server. The client has three verification server options to choose from that include UC Berkeley, Stanford University, and the University of Washington. Using the pre-computed routes between each verification server and Bank of America, the client selects the verification server with the least number of shared AS links. In this case, the optimal verification server is at Stanford University since it shares no path overlap with the connection between the UCSB client and Bank of America. Conversely, the verification server at Berkeley would be considered the least suitable of the three, since its path to Bank of America shares two AS links (CENIC and Level 3 Communications) with the path from the client to the bank.

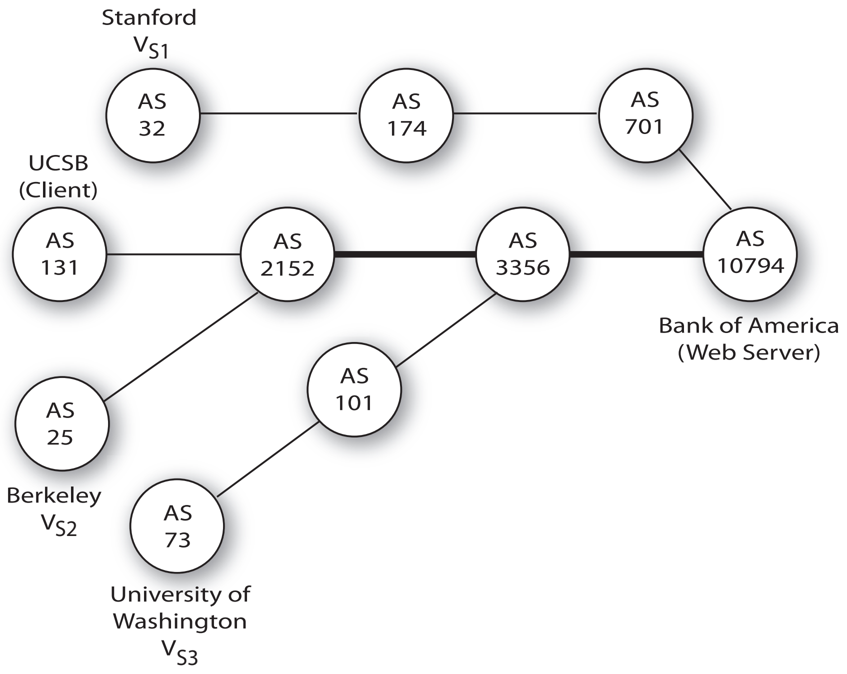


Fig. 7. Verification server selection example for a client at UCSB to the web server at Bank of America. Path overlaps are highlighted in bold.

Our path selection algorithm provides an important advantage, since the IP to AS mappings enable us to determine *a priori* the relative security of the verification process. Thus, we can ensure that the client, verification server, and web server are not on the same networks by analyzing each host's AS number (AS{C}, AS{VS}, AS{WS}). If any two have the same AS number, there is a higher risk of a man-in-the-middle since one or more links will likely be in common. In addition, the path selection algorithm that we utilize also enables us to determine the relative effectiveness of a chosen verification server. In the worst case, $(C \Rightarrow W_S \subseteq V_S \Rightarrow W_S)$, which signifies that the client and verification server share the same path to the web server and the verification process will not be effective. Furthermore, our topological maps allow us to select verification servers efficiently even when a client machine is physically relocated (*e.g.*, a laptop that may frequently change locations).

We implemented our selection process by computing the shortest paths and locations of verification servers in relation to various netblocks using AS topological mapping engines such as CAIDA's `skitter`[2], Route Views [13], and the Routing Information Service (RIS) [18]. We then correlated these maps with publicly available AS to IP address netblock mappings [14], and from the Routing Assets Database (RADb) [17]. After obtaining this data, we corroborated these paths using multiple trace routes on PlanetLab using `Scriptroute` [19]. These measurements confirmed that these paths were approximately 80% accurate with errors occurring in resolving IP addresses to AS numbers, and due to alternative AS paths. When the IP address of a host could not be resolved to an AS, we chose verification servers at random.

5 Evaluation

In this section, we analyze the performance impact of VeriKey and discuss the security of the system against man-in-the-middle attacks. We then follow with a discussion about our system's limitations and how it can be extended for enhanced performance and added security.

5.1 Experimental Setup

After implementing our system, we deployed 47 verification servers across five continents using PlanetLab. We then recorded measurements of the certificate verification process from a range of locations for the client, verification servers, and web server. The purpose of our experiments was to determine approximate bounds on the time necessary for the certificate verification process to complete. We estimated these bounds by taking geographically diverse measurements from regional, national, and multi-national locations. More specifically, the regional test involved machines across California, the national test involved machines traversing the entire United States, and the multi-national experiments entailed

[2] http://www.caida.org/tools/measurement/skitter/

Table 1. Approximate VeriKey overhead for non-cached certificates.

	Standard SSL Handshake		Certification Verification	
	Bytes Transferred	RTTs	Bytes Transferred	RTTs
C ↔ S	2,411	5x	2,411	5x
C ↔ V	-	-	2,857	7x
V ↔ S	-	-	2,411	5x
Total	2,411	5x	7,679	17x

global pathways. Each experiment consisted of a single client, verification server, and web server.

5.2 Verification Process Overhead

Before we examine the experimental measurements, we first present the overhead for a single session. Table 1 demonstrates the bandwidth overhead involved for the certificate verification process from a single verification server, which is independent of the location of the verification server. Assuming symmetric connectivity between the client, web, and verification servers (*i.e.*, the verification server is roughly half the distance between the client and web server), the round-trip-time is approximately 3.5 times a standard SSL handshake without caching. The overhead in bytes is slightly over three times the standard SSL handshake. The absolute amount of time and overhead, however, are relatively small as only about five extra kilobytes are required, and the latency for certificate verification is typically on the order of one second (more precise numbers based on the locations of the verification servers are shown in Table 2). More importantly, this process is only a one-time cost, and after the initial verification process, the SSL session resumes with no additional overhead. The round-trip-time can be further reduced by more than 50% when temporary certificate caching is enabled on the verification servers. This diminishes the need for the SSL handshake between the verification server and web server, reducing the overhead to only double that of a standard SSL handshake.

Table 2 displays the results of the verification delay for servers from various geographical locations averaged over ten consecutive trial runs. We executed two types of verification tests for each web server. In the initial test, the verification server had not previously cached the certificate of the web server. In the following test, the verification server had already cached the web server's certificate and directly returned it to the client, thereby reducing response time and subsequent connections to the web server. Each set of tests was performed ten times and the measurements were recorded. When the client, verification server, and web server were physically closer, the overall performance gain of caching (measured in latency) decreased by about 16%. This reduction in performance was due to the limit on the latency between the client and verification server.

In the approximate worst case, the verification process could take almost three seconds. While this delay would be evident to a user, we could potentially

Table 2. Verification process delay perceived by clients.

Test	Client	V_S	W_S	Verification Time	
				Non-cached	Cached
Multi-National 1	ucsb.edu (USA)	uestc1.edu.cn (China)	univie.ac.at (Austria)	2.876s	1.830s
Multi-National 2	canterbury.ac.nz (New Zealand)	u-tokyo.ac.jp (Japan)	berkeley.edu (USA)	2.303s	1.891s
Multi-National 3	mit.edu (USA)	utoronto.ca (Canada)	zib.de (Germany)	1.040s	0.691s
National	harvard.edu (USA)	colorado.edu (USA)	uci.edu (USA)	0.885s	0.674s
Regional	uci.edu (USA)	berkeley.edu (USA)	ucsb.edu (USA)	0.236s	0.204s

leverage this delay to query closer verification servers. Depending on the number and density of the deployed verification servers, the average case would most likely appear similar to the regional and national examples with verification times of less than one second.

5.3 Man-in-the-middle Attack Prevention

In this section, we examine the protective measures of VeriKey against man-in-the-middle attacks. The potential for a man-in-the-middle attack can exist on the same network as the client establishing the egress SSL connection, the ingress web server's network, or anywhere in between. Our system successfully prevents most of the possible attack vectors, and prevents all attacks that we consider likely under our set of assumptions.

The most likely location for man-in-the-middle attacks are on client networks since they are not routinely monitored and most operating systems do not implement proper safeguards. However, we can establish secure communication between the client and verification server. As previously discussed, this is a result of requiring clients to install a web browser plug-in that stores the IP addresses and public keys of the verification servers. This prevents the primary classes of attacks including a rogue DHCP server, an attacker who has poisoned the ARP caches of nearby hosts, and DNS spoofing that would redirect users to a malicious server. Because all messages are encrypted with the verification servers' public keys, only the trusted verification servers are able to decrypt the packets. If a client receives any certificate other than that of the verification server, it becomes trivial to detect malicious behavior, and block further SSL communications.

The verification servers' networks could contain a man-in-the-middle attacker. However, on the client-side, the attacker would be trivial to detect during the SSL handshake since the certificate would not match that in the client's trusted CA root. In addition, our verification server path selection algorithm will choose a verification server with the least amount of overlap between the path from the web server to the client, and the path from the web server to the verification server. Hence, a successful attack would essentially require a coordinated attack on multiple autonomous systems. A VeriKey client can also detect a compromised verification server by maintaining a history, and comparing the results from other verification servers. If a particular verification server consistently responds with invalid keys over a prolonged period of time, the client may remove the verification server from its trusted list.

In our system, we eliminate user dialog messages for the average user because we believe the messages do more harm than good. Hence, we systematically verify or reject the certificate without the need for human interaction. For advanced users, VeriKey offers additional information about the potential threat and the number of inconsistencies among the verification servers. Thus, our system provides versatility and security for both average and expert users.

5.4 System Limitations

In this section, we address the limitations of our system, which include man-in-the-middle attacks that occur on the web server's network, and the potential for a denial of service.

A man-in-the-middle can exist on the web server's network, which is the hardest attack to detect from the viewpoint of the client and verification server. However, we believe that it is also the least likely place for an attack to occur since most web servers are on networks that employ firewalls, actively monitor for malicious behavior, and employ IDS systems. The only possible way to detect an attack near the web server is to require prior registration (*e.g.*, obtaining a signed certificate from a CA).

Another potential weakness of the system is that a man-in-the-middle could potentially detect and block all traffic to or from verification servers. By creating a denial of service (DoS) attack against verification servers, the attacker would prevent a client from being able to verify any certificates. However, since it would be rather trivial to detect this type of abuse, the system would still be constructive in preventing SSL man-in-the-middle attacks (even though the client would no longer be able to communicate with the server).

VeriKey's distributed architecture also protects against DoS attacks against individual verification servers by removing any single point of failure. Therefore we maintain constant availability with a large distributed set of servers, and in the event that a verification server is attacked, it will not affect the entire system.

5.5 Security and Performance Optimizations

In this section, we analyze possible extensions to our system to provide better security and performance. As mentioned previously, a man-in-the-middle attack detected by our system may cause a denial of service for the client because the client's connection to the web server would be blocked for the duration of the attack. On the other hand, if a man-in-the-middle has not compromised the connection between the client and verification server, we may be able to use the

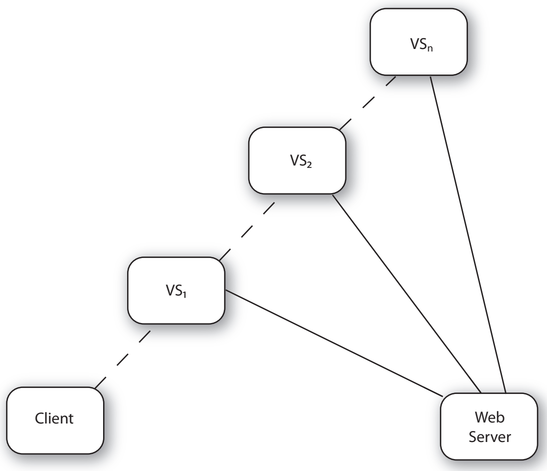


Fig. 8. Verification servers operating as SSL proxies.

proxy connection to the web server. Thus, the verification server could operate as an SSL proxy server, allowing the client to securely browse the remote web server as shown in Figure 8. In order to prevent a man-in-the-middle between the verification server and web server, the verification proxy server would function as a client web browser. That is, the verification server would follow the same process described in Section 4.4 and connect to other verification servers to ensure that it has not become compromised by a man-in-the-middle attack itself.

The major downside of this design is that it puts substantial trust into the verification proxy server and assumes that the verification server itself has not been compromised. In terms of overhead, this approach would also place a larger resource burden particularly on verification proxy servers. The client would also notice an increase in latency for the entire duration of the session rather than just the initial connection setup.

Another optimization to improve our framework's response time is to cache public keys from previous verification requests. This optimization works well, provided that the remote certificate has not recently changed. In order to determine when the verification server needs to update a certificate, we propose a method to maintain a history of client feedback during prior verification sessions. That is, when clients report a certificate mismatch, the verification servers will contact the web server to update the certificate. This would employ the knowledge gained from the optional confirmation status at the end of the VeriKey process as described earlier in Section 4.4. We introduce the concept of a reputation [4] to prevent a malicious host from deceiving verification servers into launching denial of service attacks by directing them to persistently contact a web server. This procedure would provide the ability for verification servers to identify compromised networks and construct a reputation-based scheme per IP Class C netblock, in addition to determining whether any of its own cached public keys may be outdated. Verification servers would refresh public keys from a web server only when a client with a positive reputation reported a key that did not correspond to the one stored in its own internal cache. This method serves two primary purposes: to establish that there is no man-in-the-middle between verification server and web server, and to update the verification server's public key cache when a web server's certificate changes. The downside of utilizing the reputation scheme is that it may not always be trustworthy and would require additional storage resources on the verification servers.

6 Conclusion

In this paper, we presented the design, implementation, and evaluation of an architecture to augment the security of SSL public key exchanges. In particular, we have introduced a means to verify the integrity of self-signed certificates. For average clients, we have substituted user interaction with an automated technique to utilize the views of remote peers, while empowering expert users with supplemental information about public key exchanges to make better assessments when accepting certificates.

We have shown the benefits, performance impacts, and the limitations of our system. Although our system is not free of weaknesses, we believe that the advantages that it provides far outweigh the potential drawbacks. Through our analysis, we are confident that our approach significantly increases the level of difficulty for a miscreant to launch a successful attack. As more insecure wireless networks are deployed, the number of attacks will likely increase because these networks provide a more susceptible environment in comparison to traditional wired LANs. Therefore, we see the need for a cost-effective and lightweight solution, such as the one that we have proposed, to protect and prevent users from becoming victims of these attacks.

References

1. P. Bahl, A. Balachandran, and S. Venkatachary. Secure Wireless Internet Access in Public Places. In Proc. of the International Communications Conference (ICC), Helsinki, Finland, June 2001.
2. D. Bruschi, A. Ornaghi, and E. Rosti. S-ARP: A Secure Address Resolution Protocol. In Proc. of the Annual Computer Security Applications Conference (ACSAC), Las Vegas, NV, December 2003.
3. P. Burkholder. SSL Man-in-the-Middle Attacks. The SANS Institute, February 2002.
4. V. Cahill, B. Shand, and E. Gray. Using Trust for Secure Collaboration in Uncertain Environments. Pervasive Computing, 2(3):52-61, July-September 2003.
5. T. Chomsiri. HTTPS Hacking Protection. In Proc. of Advanced Information Networking and Applications Workshops (AINAW), Mahasarakham, Thailand, May 2007.

6. J. Demerjian, A. Serhrouchni, and Mohammed Achemlal. Certificate-based Access Control and Authentication for DHCP. In Proc. of the International Conference on E-Business and Telecommunication Networks (ICETE). Setubal, Portugal, August 2004.
7. DSniff. Website: http://monkey.org/˜dugsong/dsniff/.
8. D. Eastlake. Domain Name System Security Extensions. RFC 2535, March 1999.
9. Ettercap. Website: http://ettercap.sourceforge.net/.
10. M. Gouda and C. Huang. A Secure Address Resolution Protocol. In the Computer Networks Journal, 41(1):57-71, January 2003.
11. T. Komori and T. Saito. The Secure DHCP System with User Authentication. In Proc. of Local Computer Networks (LCN), Washington DC, November 2002.
12. W. Lootah, W. Enck, P. McDaniel. TARP: ticket-based address resolution protocol. In Proc. of the Annual Computer Security Applications Conference (ACSAC), Tucson, AZ, December 2005.
13. D. Meyer. University of Oregon Route Views Project. Website: http://www.antc.uoregon.edu/route-views/.
14. Z. Morley Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-Level Traceroute Tool. In Proc. of the Special Interest Group on Data Communication (SIGCOMM), Karlsruhe, Germany, August 2003.
15. PlanetLab: An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services. Website: http://www.planet-lab.org/.
16. L. Poole and V. S. Pai. ConfiDNS: Leveraging scale and history to improve DNS security. In Proc. of Third Workshop on Real, Large Distributed Systems (WORLDS), Seattle, WA, November 2006.
17. Routing Assets Database (RADb). Website: http://www.radb.net/.
18. Routing Information Service (RIS). Website: http://www.ripe.net/ris/ris-index.html.
19. N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility. In Proc. of the Internet Technologies and Systems (ITS), Seattle, WA, March 2003.
20. R. Wagner. Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks. The SANS Institute. August 2001.
21. H. Xia and J.C. Brustoloni. Hardening Web Browsers Against Man-in-the-Middle and Eavesdropping Attacks. In Proc. of the 14th International World Wide Web (WWW) Conference, Chiba, Japan, May 2005.
22. L. Zhou, F. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. ACM Transactions on Computer Systems, 20(4):329-368, November 2002.