Machine Learning and Artificial Intelligence
A.J. Tallón-Ballesteros and C. Chen (Eds.)

doi:10.3233/FAIA200789

Secure Server Key Management Designs for the Public Cloud

Kevin FOLTZ[1] and William R. SIMPSON
Institute for Defense Analyses, 4850 Mark Center Drive, Alexandria, VA 22311

Abstract. The Enterprise Level Security (ELS) model focuses on designing secure, distributed web-based systems starting from basic principles. One area of ELS that poses significant design challenges is protection of web server private keys in a public cloud. Web server private keys are of critical importance because they control who can act as the server to represent the enterprise. This includes responding to requests as well as making requests within the enterprise and to its partners. The cloud provider is not part of this trusted network of servers, so the cloud provider should not have access to server private keys. However, current cloud systems are designed to allow cloud providers free access to server private keys. This paper proposes design solutions to securely manage private keys in a public cloud. An examination of commonly used approaches demonstrates the ease with which cloud providers can currently control server private keys. Two designs are proposed to prevent cloud provider access to keys, and their implementation issues are discussed.

Keywords. enterprise, security, public cloud, key management, hardware security module, web server, cloud computing

1. Introduction

Web security has become more important as more aspects of business, government, and everyday life are put online. A challenge that occurs at the design level is securely managing the private key of a web server while utilizing the public cloud. The goal is a server key management design in which keys cannot be duplicated, keys can only be used by authorized individuals, and key operations are timely. At the time of this writing, no known solution has been implemented to provide all of these. This presents a problem that affects all public cloud hosted web servers.

This paper presents a review of common existing solutions' security shortcomings, an analysis of improvements to these solutions, and two proposals for a secure design.

2. ELS in a Private Cloud

The approach to security in this paper is based on Enterprise Level Security. Many security solutions patch together components that were developed independently, which leaves legacy and insecure protocols and interfaces as easy attack points [1]. Others use third parties to provide security as part of another non-security service [2],

[1] Corresponding Author, email: kfoltz@ida.org

which puts security in the hands of non-professionals and leads to easy security flaws. Still others use third parties that assume privileged positions in the architecture to perform security-specific functions [3]. The Zero Trust Architecture is a new approach, and ELS is an architecture that is consistent with ZTA ideas and has moved toward implementation. As such, it holds promise for enterprise security.

ELS enumerates a number of tenets and concepts that guide its requirements and implementation details [4]. One important idea in ELS is that all communication and sharing is done with end-to-end integrity and encryption between two endpoints. No other entities are allowed to view or modify such communication. Also, when two entities communicate, they have strong assurance that they are communicating with each other. This relies on strong end-to-end authentication of both entities [5] and hardware for storing and using private keys for high security applications.

The ELS private cloud hosting approach is shown in Fig. 1, where a normal request flows from a browser on an enterprise machine to a server hosted in the private cloud on a virtual machine. The browser uses the smart card certificate and private key to authenticate through TLS to the server. The smart card private key is accessed through a universal serial bus (USB) smart card reader, and the smart card is protected by a personal identification number (PIN). The server uses its certificate and HSM key to authenticate to the browser. The connection from the server to the HSM is secured through a separate TLS connection. This HSM connection may be local and long-lived. Session keys are used for the encryption, decryption, and validation of TLS transmitted data.

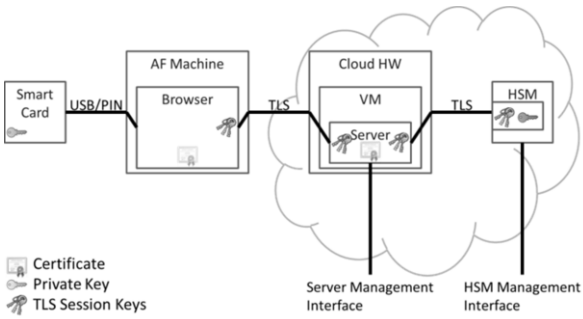

Figure 1. Normal web request flow.

3. The Public Cloud Challenge

The simplest method to move to the public cloud is to directly apply the private cloud model to the public cloud. For higher security, an administrator from the enterprise can visit the cloud provider and generate hardware key pairs on a validated HSM. A further security measure is to create only direct connections between cloud servers and the HSM and eliminate the use of virtual HSM interfaces. A final security measure is for an administrator to create all of these direct server-to-HSM connections while visiting the cloud provider.

Using all of these security measures, an administrator can create long-lived TLS connections between the servers and the HSM. Special tokens or keys that are used to create HSM connections are held by the enterprise and not shared with the cloud

provider. This prevents the cloud provider from creating additional connections. The server private keys are in hardware, and the only connections to the HSM are from the associated servers.

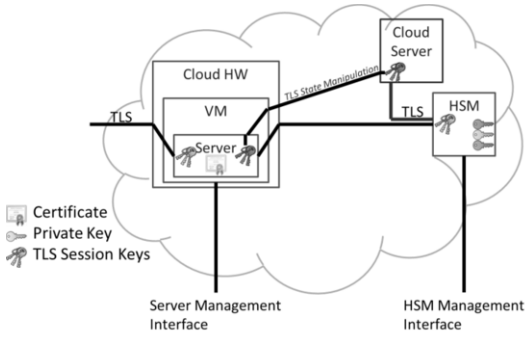

Figure 2. Web request flows in an untrusted cloud – direct TLS state manipulation to allow additional HSM connections.

However, the cryptographic keys for the secure HSM connection, along with any other necessary information including sequence numbers and other internal states, are stored in software on the server as part of the TLS connection state. This sensitive information can be extracted by an untrusted cloud provider by either copying the virtual machine on which the server is running or probing it through hypervisor interfaces. With access to the memory of the machine, methods exist to directly extract TLS keys from an executing application [6]. With these keys and the appropriate state information, the cloud provider can inject new private key usage requests to the HSM by creating the proper messages and updating the internal state of the server TLS connection, such as sequence numbers, initialization vector values, and other encryption state. This is shown in Fig. 2.

It requires specialized skills to probe a virtual machine image and manipulate TLS session keys, but the cloud provider has the required level of access to perform these actions. Even with dedicated effort to hide the TLS state in the virtual machine, the protection is only obfuscation, and such an approach fails to satisfy ELS security principles.

4. Proposed Secure Solutions

The main problems discussed in this paper have stemmed from the separation of the server and HSM and the challenge of establishing a secure connection between them. The approaches that follow attempt to combine the server and key.

4.1. Server in HSM

One way to combine server and keys is by implementing the entire server and its keys inside the HSM. The HSM server connects with the browser through a TLS connection, as shown in Fig. 3. The application is pre-loaded onto the HSM, associated with the proper private keys, and then shipped to the cloud provider. The private key access occurs completely within the HSM, which protects it from the cloud provider.

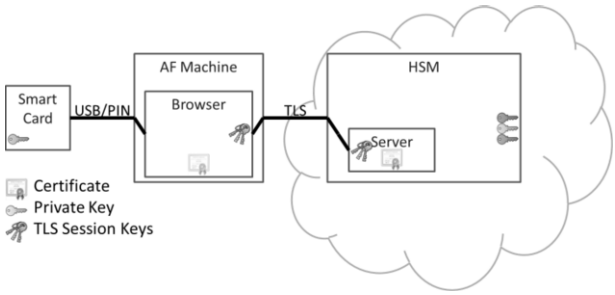

Figure 3. Proposed solution with server in HSM.

The main hurdle for this approach is that HSMs are expensive, special purpose devices designed specifically for key generation, storage, and use. They are not designed to run arbitrary software, and they typically do not have the storage or computation power required to support a full server. These are problems, but they are implementation issues within a fundamentally secure design.

4.2. Homomorphic Encryption

A second way to combine server and key is through the use of homomorphic encryption. Homomorphic encryption allows data to be processed while remaining encrypted [7]. The programs that normally operate on the unencrypted data are transformed to operate on the encrypted data.

In a homomorphic encryption solution, the requester encrypts its requests using homomorphic encryption. These requests are not decrypted at the transformed server. They are used as encrypted inputs to a program that operates on encrypted data. Part of the server processing involves authentication using an HSM. The HSM is also transformed to operate on encrypted data. The server sends encrypted key operation requests to the HSM and uses the encrypted response to authenticate to the requester. Similarly, the server can authenticate as a requester to other servers that accept encrypted requests. The design for the client to server communication is shown in Fig. 4.

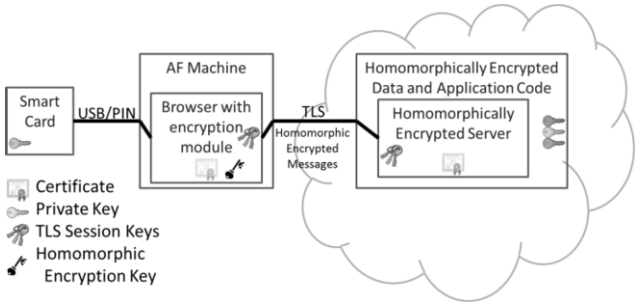

Figure 4. Proposed solution using homomorphic encryption.

The primary issue with the full homomorphic encryption (FHE) approach is performance. Current implementations of FHE are extremely slow [8]. FHE has seen significant performance improvements, but it is still far from practical.

5. Operational Considerations

The previous section discusses designs for server private key security in the public cloud. This section examines implementation issues for the secure designs.

5.1. Cloud Vendor Support

The cloud vendor controls the HSM and server and is centrally positioned to provide server private key security. Current cloud providers offer cloud-based key management services backed by HSM key storage, but they offer virtual HSM interfaces, and controls on key access are set at this virtual interface [9]. The cloud provider allows the enterprise administrator to set permissions on key use, but these permissions apply only to other enterprise users, not to the cloud provider itself. The cloud provider maintains direct control of the HSMs.

One promising approach for server integrity is a method to establish software root of trust unconditionally on hardware [10]. This provides assurance that no malware is present on a machine when it boots. The cloud provider performs this procedure to ensure that no malicious code is hidden on the hardware during the boot process. However, this still relies on the cloud provider to perform the procedure properly. Server private key security must come from outside the cloud provider.

5.2. HSM Vendor Support

An HSM can provide the property that a key cannot be duplicated. Major HSM vendors also provide some accommodations to facilitate the move to the cloud [11]. Thales, for example, provides a method to create a key on a local HSM and securely copy it to an HSM in the cloud [12]. However, this feature does not protect the cloud HSM from the attacks on the server-to-HSM connection. It also explicitly creates a copy of the private key, which is against ELS principles.

To improve security, the HSM vendor must provide a way to authenticate the server connections. However, the HSM is the key holder for server authentication, so this becomes a bootstrapping problem. Thus, the HSM vendor cannot solve this issue using advanced key management.

5.3. Leveraging Mobile Device Management (MDM) Technologies for Cloud Assets

Mobile device vendors embed hardware security features to enable secure communication with their devices. For example, the vendor can embed their own PKI root certificate authority (CA) public key in the hardware to enable trusted software updates [13]. The MDM services integrate with these hardware protections. The operating system, securely loaded by the hardware after signature validation, provides a set of APIs that can be used to remotely control the operation of the device.

The hardware protections of a mobile device make the device much like an HSM. The ability to remotely manage the phone, and in particular the ability to restrict user functions, makes such a device desirable for cloud hosting environments. In this scenario, the cloud provider acts as the mobile user, and the cloud client acts as the MDM administrator. The client hosts their server on a cloud-provided mobile device and manages it in the potentially hostile cloud environment through the MDM

interfaces. This allows the remote client to limit the activities of the hands-on cloud provider instead of the cloud provider limiting the client's activities.

6. Conclusions

The public cloud offers many desirable benefits in cost, efficiency, and even security, but taking advantage of these while maintaining secure server private key management is a challenge. Problems in the public cloud and hybrid cloud stem from the separation of the server from its authentication key in the HSM. An untrusted cloud provider can exploit this separation to use the private key of the server.

The proposed solutions for secure design have the common theme of encapsulating the server and HSM within a single logical entity. This eliminates the difficult problem of authenticating the server and HSM to each other. Using an HSM as the single logical entity provides a hardware encapsulation that hides the communications between the server and key management within the HSM itself. Using homomorphic encryption creates this single logical entity in software by encrypting the requests and responses and allowing the cloud provider to access and process only this encrypted data.

Current technology appears poised to be able to implement these approaches. The performance of FHE is currently poor, but it is improving. Mobile devices include technology that can allow secure remote management and software updates, which is similar to what is required of the "server in HSM" solution.

References

[1] Ming Lu, technical discussions regarding IBM Tivoli, July-December 2014.
[2] Farhan Saifudin, technical discussions regarding Mobile Iron mobile device management security, June 2018.
[3] Jonn Inscoe, technical discussions regarding F5 web application firewall, F5 Government Technology Symposium, Washington, DC, December 2014.A.N. Author, Article title, Journal Title 66 (1993), 856–890.
[4] Kevin E. Foltz and William R. Simpson, "Enterprise Level Security – Basic Security Model," 7th International Multi-Conference on Complexity, Informatics, and Cybernetics: IMCIC 2016, Orlando, Florida, March 2016.
[5] Coimbatore Chandersekaran and William R. Simpson, "The Case for Bi-lateral End-to-End Strong Authentication," World Wide Web Consortium (W3C) Workshop on Security Models for Device APIs, 4 pp., London, England, December 2008.
[6] Brendan Dolan-Gavitt, T. Leek, J. Hodosh, and W. Lee, "Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection," Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2013.
[7] Craig Gentry, 2009. "A Fully Homomorphic Encryption Scheme." Doctoral thesis. Stanford University.
[8] Virgil D. Gligor, "Homomorphic Computations in Secure System Design," Final Report. Pittsburgh, PA: Carnegie Mellon University, 2014.
[9] Amazon Web Services (AWS), "AWS CloudHSM." Available at https://aws.amazon.com/cloudhsm/.
[10] Virgil Gligor and Maverick Woo, "Establishing Software Root of Trust Unconditionally," Network and Distributed Systems Security (NDSS) Symposium 2019, February 24-27, 2019, San Diego, CA, USA, https://dx.doi.org/10.14722/ndss.2019.23170.
[11] Personal discussions with representatives of major HSM providers, RSA Conference, March 4-8, 2019, San Francisco, California.
[12] Personal discussions with representatives of major HSM providers, BlackHat, August 7-8, 2019, Las Vegas, Nevada.
[13] Apple Inc., iOS Security, iOS 12.3. May 2019. Available at https://www.apple.com/business/docs/site/iOS_Security_Guide.pdf