Accessing Cloud through API in a More Secure and Usable Way

HongQian Karen Lu

Gemalto, Inc. Austin, Texas, U.S.A.

Abstract. A common method for accessing and managing cloud computing resources is through an Application Programming Interface (API). Each API request from an application must include a client authentication to the cloud service, which proves the possession of a secret key. Securing such keys is critical to the confidentiality, integrity, and availability of the data and services hosted in the cloud. Currently users manually handle these keys; a process that is neither secure nor user-friendly. Where to store the keys and how to access them are still security challenges especially for those applications that reside in the cloud themselves. Furthermore, keys are in clear text at least in a computer's memory. Attackers can find ways to recover them. This paper presents a solution to these problems by using portable security devices. The device securely exchanges keys with the cloud serve, securely stores the keys, and performs cryptographic computations using these keys for the client authentication. The user must have the device and authenticate to it in order use it. The solution enables a two-factor hierarchical security protection of the cloud computing resources. It not only enhances the security but also improves the usability.

1 Introduction

Cloud computing enables an on-demand network access to a shared pool of configurable computing resources. It provides scalability, flexibility, and fault resilience. Security is a key issue in outsourcing computing and data to the cloud. The confidentiality, integrity, and availability of data and applications are crucial; any breach would results in financial losses, reputation damage, loss of compliance, and pain to the customers. To mitigate these risks, cloud providers deploy layered security, including physical security, network firewalls, separation of virtual machines, and access controls.

In addition to web portals for service accesses, cloud providers publish Application Programming Interface (API) for client applications to programmatically access and manage cloud computing resources, such as provisioning, operating, and monitoring. The client application may be a web application in a cloud, a web server within an organizations firewall, or a desktop software in a user's computer. To secure the API-based cloud access the client application must include authentication information in each API request, which proves the possession of certain secret. For example, some cloud API requires a digital signature for each request. Such client authentication serves dual purposes: access control and accounting.

Karen Lu H..
Accessing Cloud through API in a More Secure and Usable Way .
DOI: 10.5220/0003559100250038
In *Proceedings of the 8th International Workshop on Security in Information Systems* (WOSIS-2011), pages 25-38
ISBN: 978-989-8425-61-4

For API-based cloud access, typically the cloud provider generates one or more secret keys or key pairs. These keys are credentials that client applications use for authenticating API requests. Therefore, secure handling of these keys is critical. This presents many challenges, including the following:

1. Key downloading and storage. The user copies the keys from the cloud provider's website and stores them. This posts security risks as the keys are in clear text and the user's computer may have malware. Usability is also an issue due to the cumbersome manual handling of the keys.

2. Key usage. To access the cloud, a client application loads a key from a user specified location; the key may be hardcoded in the application; or the user may send the key back to the cloud server [20]. The security and usability issues are similar as that in #1.

3. Client in cloud. The case that the client application is running in a cloud itself is especially challenging: how to load the key to that application and where to keep it [10]?

4. Key in memory. Even if one encrypts the key before storing it to a disk or a device, the key has to be decrypted before use and it is in clear text inside the computer's memory. If there is a malware, an application is compromised, or an attacker gets hold of the computer, he can recover the key.

5. Server side vulnerability. When the cloud website allows downloading keys, vulnerabilities, such as cross-site scripting, in the website may enable attackers to steal the keys [9].

A broker service was proposed to manage the cloud connection and the access keys [23]. However, one still faces the problems of securely accessing such a service and loading keys to it.

This paper presents a solution to the above problems. We propose using portable security devices (PSD) to manage the keys and to secure the API-based access to the cloud. Figure 1 illustrates two examples of PSDs, one is a credit- card-shaped smart card, and another is a secure USB token with an embedded smart card.

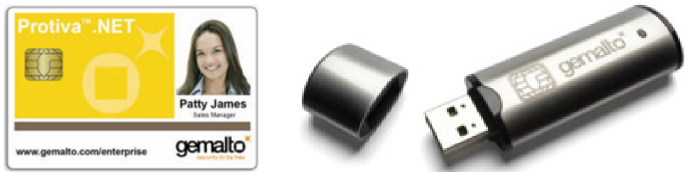


Fig. 1. Smart-card-based portable security devices.

Desktop applications and mobile devices have used smart cards for user authentications [12]. Amazon Web Services (AWS) offers a multi-factor user authentication for account login [2], which is based on the One-Time Password (OTP) [14] hardware token. We extend the PSD's services to the API-based cloud access to address the security and usability issues outlined earlier. This goes beyond the user authentication and presents a different set of challenges. For example, how to exchange keys between a cloud provider and a PSD? How to associate keys in the PSD with cloud resources? How to use the PSD during the formulation of API calls? How to make this security solution user-friendly as well? This paper describes our answers.

A computing cloud typically has a hierarchical structure that organizes its resources, such as services and virtual machines [1]; virtual data centers, clusters, and storage volumes [17]. We generalize the idea of using a PSD to the cloud structure. This enables configuring the level of protection, usability, and performance based on needs.

Our method provides a two-factor security protection to the cloud computing resources: it requires the user to know something (e.g. PIN) and to possess something (a PSD). Using the method, no secrets are transferred unencrypted, private keys never leave the PSD, and no keys will be clear text in a user computer's memory or disk. This significantly improves the security. Furthermore, the users do not manually handle the keys, which improves the usability as well. As a result, the method helps to mitigate risks associated with three out of the seven top threats to cloud computing [6] (insecure interfaces and APIs, data loss or leakage, and account or service hijacking).

We have applied existing technologies to solve new real world problems in cloud computing and made the cloud access more secure and user-friendly. The contributions of this paper include the following:

- A method for securing API-based cloud resource access using PSD.
- Techniques for establishing and exchanging keys between the cloud service provider and PSD, and associating keys with cloud resources.
- A method for enabling hierarchical protection of cloud resources.

The rest of the paper is organized as follows. Section 2 outlines related work. Section 3 introduces the system model, the threat model, and portable security devices. Section 4 presents the basic operations. Section 5 describes the architecture of the two-factor hierarchical security protection method. Section 6 presents a proof of the concept. Section 7 discusses how our method solves the problems listed in this section, and outlines future works.

2 Related Work

Security is a major consideration for the adoption of cloud computing services. The IEEE and the Cloud Security Alliances (CSA) have collaborated to develop security standards and to promote best security practices. The CSA has published the security guide [5] and identified seven top threats to cloud computing [6].

Cloud providers, such as Amazon Web Services [1], Microsoft Windows Azure [13], Rackspace Mosso [18], and Terremark [22] publish APIs for applications to programmatically access the cloud resources. Most APIs are RESTful web service APIs.

The cloud API typically requires a client authentication. For examples, AWS Query API requires computing HMAC-SHA hash using a secret access key [1]; Windows Azure API requires SSL mutual authentication using the client X.509 certificate [13]; Mosso API requires an authentication token for accessing cloud services [18]; and Terremark API uses HTTP basic authentication [22]. Oracle recently proposed a cloud management API as a standard to the Distributed Management Task Force [7]. The proposal also specifies the HTTP basic authentication for each API request [16]. The Open Cloud Computing Interface [15] is a set of open specifications from the Open Grid Forum. Unfortunately, OCCI only recommends that a client authentication mechanism be used where appropriate.

Cloud providers tell the users to keep the secrets well protected and make it clear that it is the users' responsibility to do so. However, users do not have good tools to use and, hence, typically manage the secrets manually, which is not secure. There are practical tips for software developers to mitigate the security risks associated with the keys [10]. Each of these tips has pros and cons, but no panacea [21]. Another possible solution is to use a broker service to manage the access keys and to secure the connection to the cloud [23]. However, the user or the application still faces the problem of how to access such as a service securely.

The client API request authentication is different from the user authentication. The client application needs to include the proof of the possession of a secret for each API request. As a result, it either keeps the secret in memory or gets it from somewhere each time. The user typically authenticates once for each login session to access all the services. As a result, the application does not need to remember or even know the user's secret, unless the application caches the user login credential for the next session. Due to these differences, client authentications face different challenges and, hence, need new technologies, such as the one presented in this paper.

The most commonly used method of user authentication is username and password, which is also the weakest and susceptible to many attacks. Cloud providers are working on enhancing the user authentication process [3]. For example, Amazon Web Services gives users an option for the multi-factor authentication (MFA) using a One-Time-Password (OTP) hardware token [2].

This paper focuses on securing cloud access through APIs and securing the keys used for such an access. User authentication is outside the scope of this paper, even though the same PSD can also serve as a user authentication token.

Banks in some countries allow online users to digitally sign money transactions using their smart cards. Conceptually, signing cloud API requests using PSD is similar to signing online banking transactions. Practically, the former is more generic. It applies to API requests for cloud resource access, and is not limited to specific applications. Furthermore, our contributions also include methods for key exchange, key-resource association, and hierarchical resource protection.

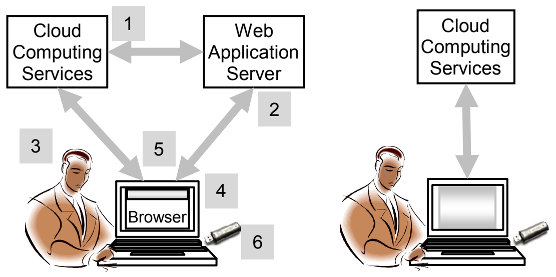
(Note: for simplicity the term key is used when there is no need to distinguish a secret key or a private key.)

3 Problem Formulation

This section presents the system models and the threat model that this work focuses on. We also briefly describe the properties of portable security devices (PSD) and explain why they are the choice of use.

3.1 System Models

We focus on the two commonly used models for the cloud access: web application and local applications, both of which are clients of the cloud servers. (See Figure 2.)



a. Web model b. Local model

Fig. 2. The system models.

The web application model can be further classified into two models: (1) application running in a cloud and (2) application running within an organization's domain, most likely behind the firewall. The later is called a hybrid model, which leverages existing computing resources and uses the cloud resources as needed. It allows the organization to keep its key assets in house for better control. It also allows an incremental approach toward moving to the cloud.

The web system model consists of the following:

1. A cloud server providing computing resources and API;
2. A web application;
3. A user (e.g. account administrator) working through a web browser to interact with the web application or the cloud website directly for accessing the cloud resources.
4. A user's computer;
5. The web browser, running in the user's computer;
6. A user's portable security device (PSD).

With the local model, the cloud client application runs inside the user's computer or inside the web browser. The user interacts with the local application or the web browser to access his cloud computing resources.

For both models, the software running in the web browser or in the computer can communicate with the PSD. All connections between the software running in the user computer, the web application server, and the cloud services are TLS/SSL. The cloud service requires the proof of the possession of a secret key or a private key for every service request through the API.

3.2 Threat Model

We assume that the cloud services and the PSD are trustworthy and secure; and that the user is honest. We do not fully trust the user's computer, which may contain malware.

Typically, the cloud provider generates an access ID and a secret access key when the user signs up for a cloud account. The user copies the ID and the key after logging into the account, and stores them. Afterwards, the user can request generating more credentials with symmetric or asymmetric keys and download the keys. An X.509 certificate may be generated by the cloud provider or uploaded by the user. In the former case, the user downloads the associated private key from the cloud server.

There may be malware in user's computer or web browser; there may be vulnerabilities in applications. In these cases, an attacker may steal the key during the key downloading, storing, or usage, since the keys are in clear text on the user's computer, at least in the computer memory. If the client is a web application in a cloud, the key needed to access cloud services through API may also be vulnerable regardless of how it is transferred and if it is stored in memory, on a disk, or not stored [21]. This paper addresses these problems. The malware or attackers may do other damages; the user may perform inappropriate operations intentionally or unintentionally. These problems are outside the scope of this paper.

3.3 Portable Security Devices

We focus on smart-card-based portable security devices (PSD). A smart card is a small card that contains a secure and tamper-resistant microprocessor chip. It has secure memories, performs cryptographic computations, and contains security applications. Smart cards are used successfully as subscriber identification modules (SIM) in cell phones, EMV cards for banking [8], and cards for computer access, digital signature, and so on.

A PSD requires a PIN to use. A user needs to possess the PSD and knows the PIN. This provides a two-factor security protection. The PSD typically has a try counter to prevent brute-force attacks. The try counter is normally set to a small number, for example 3. For each wrong PIN entry, the PSD decrements the try counter by one. The PSD locks itself if the user has exhausted the try counter.

A user can enter the PIN from the computer either via a keyboard or a virtual PIN pad; or by using an external secure hardware PIN pad. The hardware PIN pad is much more secure because the PIN never enters the computer.

4 Core PSD Functions

We propose a two-factor security protection method for API-based cloud access. The two factors are what you know (e.g. PIN) and what you have (a PSD). The PSD contains the keys for the client application to access cloud resources through the API. The user must authenticate to the PSD to use it. This section describes core functions that the PSD should perform to support this method.

4.1 Secure Key Storage

The PSDs have a tamper-resistant secure storage that can store keys and other sensitive information. Since the device itself requires user authentication, an attacker would need both the device and the PIN to steal or damage the cloud resources. Due to many logical and physical security measures built into the smart cards [24], it is prohibitively difficult and expensive to steal keys from the cards even for experienced hackers with sophisticated equipment.

4.2 Secure Key Exchange

Before issuing a PSD to a user, the issuer can provision the PSD with an X.509 certificate and the associated private key for communicating with the cloud provider. For key exchange, the PSD sends its certificate to the cloud provider to associate the certificate with the user account. The cloud server can then encrypt the secret access key using PSD's public key and send the encrypted key to the PSD.

Alternatively, the PSD can generate and store a key pair or a secret access key. The device sends the public key to the cloud server, or encrypts the secret key using the server's public key and sends the encrypted secret key to the server.

4.3 Cryptographic Computations

Some clouds APIs require cryptographic computations, such as asymmetric-key-based digital signature or keyed cryptographic hashing, for the client authentication in each service request. Such computations are called signing the API [1]. The result is called a signature. The key (private or secret) used for the computation is called the signing key. Note: if a cloud API simply requires a password for client authentication, we may consider the password as the signature and the signing operation is basically getting the password.

We propose using PSD to perform these cryptographic computations. In this way, the secret or private keys remain inside the device, preventing attackers from stealing them. The device does not provide any interface to export the private key nor to export the secret key without encryption.

5 Architecture

The two-factor security protection method uses the core functions described in Section 4. It further includes procedures for device registration, association, and resource connection. The (hierarchical) relationships of the resources in the infrastructure enable the cloud providers and users to configure the security, usability, and performance based on their needs.

5.1 General Operations

The user first registers the PSD with his cloud account. The resource protection has two phases: association and connection. The association relates a key (pair) identifier with a resource; the connection lets the client application connect to or access the resource.

5.1.1 Registration

The user can register the PSD during the account activation or afterward. The registration has the following steps. (See Figure 3).

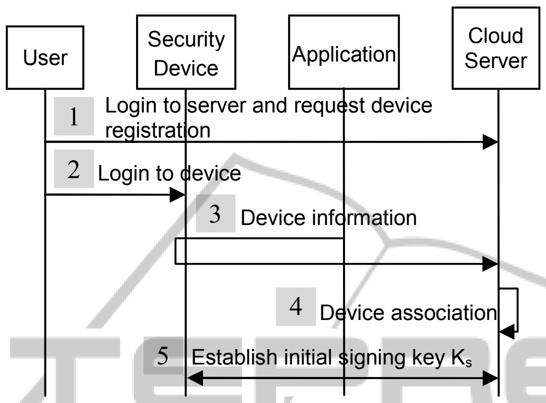


Fig. 3. Device registration process.

1. User logs into his cloud account, or he creates an account through an application. He requests to register his PSD.
2. User logs into his PSD, using PIN, for example.
3. At User's request, the client application gets the PSD information, such as the serial number and public key. It sends the information to the cloud server.
4. The cloud server associates the PSD information with the account.
5. The cloud provider may want the PSD to use its private key as an initial signing key, or to use a shared secret as the signing key using the method described in Section 4.2 for key exchange. The PSD uses its signing key to sign initial requests to the cloud server before other keys are established.

5.1.2 Association

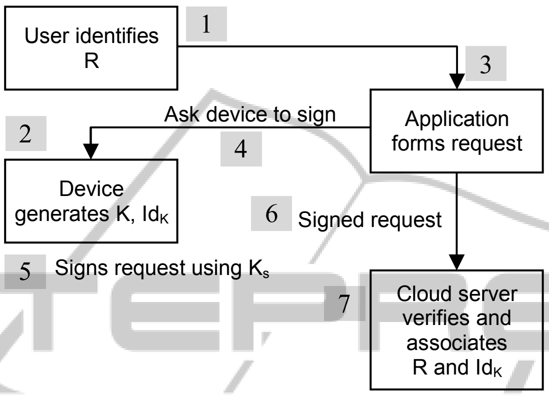
The association phase creates a key pair or a secret key and associates the key (pair) identifier with a resource in the cloud.

Let K be the key (pair) for the resource R; Id_K be the identifier for K; K_S is the signing key established in the registration phase. The goal is to associate Id_K with R. The PSD uses the signing key K_S to sign the request to the cloud server for this association. Once the association is established, the PSD uses K to sign all the future requests for the resource R.

Figure 4 illustrates a high level view of the association process. It goes as follows:

1. User identifies the cloud resource R.
2. User requests the security device to generate a key (pair) K. The device generates K and Id_K, and stores them in its secure storage.
3. The client application creates the request for associating R and Id_K.
4. The client application asks the device to sign the request.

5. The device signs the request using its signing key K_S, and returns the result to the client application.
6. The client sends the signed request to the cloud server, which includes the public key of K or encrypted K.
7. The server verifies the request, for example verifies the signature and associates R with Id_K.



1
User identifies R
3
Application forms request
Ask device to sign
2
4
Device generates K, Id_K
6
Signed request
5
Signs request using K_s
7
Cloud server verifies and associates R and Id_K

Fig. 4. Key association process.

5.1.3 Connection or Service Request

The connection phase allows the user to connect to a resource R or to make a service request for R using the associated key K stored in his PSD. As mentioned earlier, the request for R must be signed using K. Figure 5 illustrates a high level view of the connection or service request process.

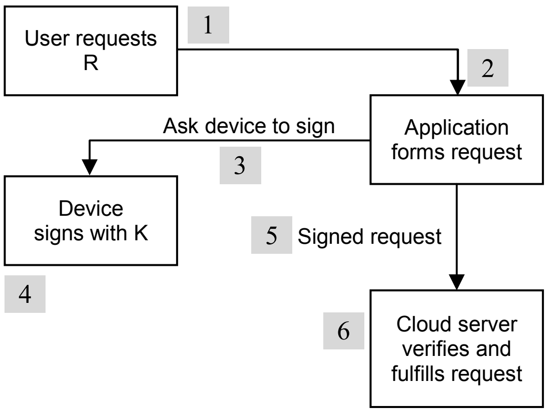


Fig. 5. Service request process.

The service request process involves the following steps.
1. The user wants to make a service request for resource R.
2. The client application prepares the request.

3. The client application asks the security device to sign the request. There are various ways for the device to know which key to use. For example, if R is at the account level, the client asks the device to sign using the account signing key $K=K_S$. Otherwise, either the user selects Id_K, the client finds it from the server and tells the device, or the device keeps the (R, Id_K) pairs and the client tells R to the device.
4. The device performs the operation using K that Id_K represents, and returns the result.
5. The client application sends the signed request, which includes R, to the server.
6. The server verifies signature and provides the service or connection.

5.2 Hierarchical Security Protection

Cloud providers adopt various hierarchical structures for organizing cloud computing resources [1]; [17]. Some cloud providers allow the user to use the account credentials to access all the resources belonging to the account; others enable the user to have a finer security control of the resources, for example, associating a unique key pair to a virtual machine instance. (See Figure 6).

Our two-factor security protection method can facilitate the access control in the cloud's hierarchy. Different methods exist for authenticating API requests. There may be more than one type of API with different authentication methods and different kinds of keys to access a same resource. Associating various keys and methods with the hierarchical resource structure enables finer access controls but makes the implementation, key management, and usage more complex. Using the methods described in the previous sections, a PSD helps to manage this complexity. The device can store multiple keys and perform various cryptographic operations, which makes it possible to support multiple access control methods to various resources in the cloud hierarchy.

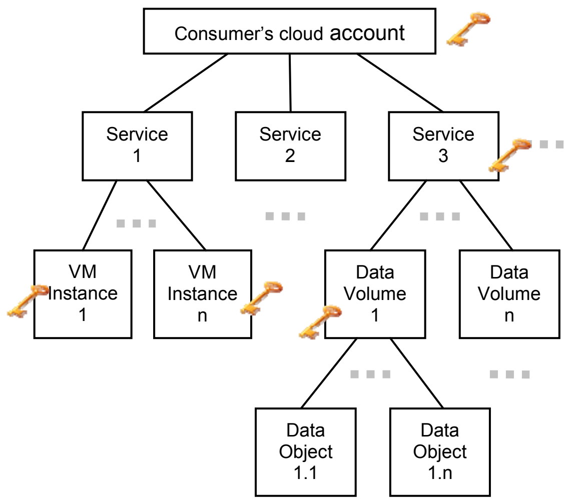


Fig. 6. Hierarchical structure of the resources.

The PSD-based approach also allows the user or the cloud provider to customize the security level based on the needs in security, performance, and usability, or the balance of them. For example, (1) the granularity of the security protection: some resources have their own keys, while others may inherit or share keys; (2) usage of one or multiple PSDs for different cloud services; and (3) single or multiple user authentications for various requests.

6 Proof of Concept

We have implemented the core technology presented in this paper. A full implementation requires the cloud provider's collaboration. Our system uses the web model illustrated in Figure 2-a. It implements the PSD-based client authentication for accessing a commercial public cloud. The PSD is a Gemalto.Net Smart Card [11]. We use SConnect [19] for the web application to communicate with the smart card. The card stores a secret access key generated by the cloud provider. For each cloud service request, the web application asks the smart card to sign the request and includes the signature in the API request. The web application has two parts, a server and a client. The server serves the client and communicates with the cloud service. The client runs inside the user browser.

For example, the user wants to monitor his virtual machine instances hosted at the cloud provider. To fulfill this request, the client asks him to insert the smart card, and then to enter the PIN. (See Figure 7.)

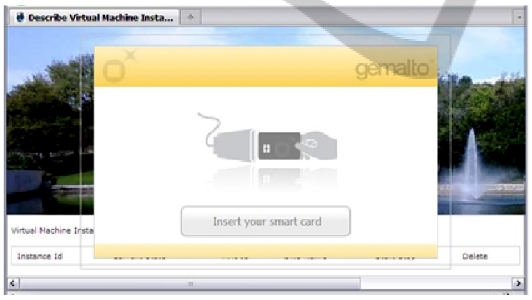


Fig. 7. The user must insert the smart card to continue.

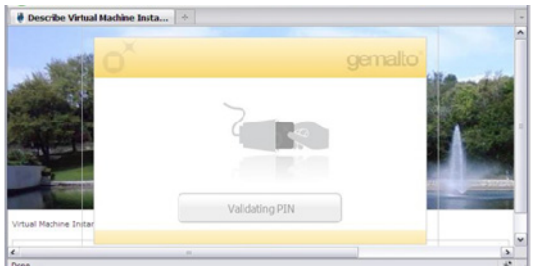
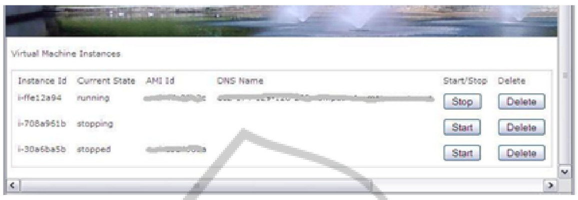


Fig. 8. The smart card is validating the user PIN.

The smart card verifies the PIN. (See Figure 8.) If it is successful, the client asks the card to sign the service request that it has received from the application server. The card signs the request and the client returns the signature to the server. The latter sends the signed request to the cloud server, which returns a list of virtual machine instances. The client displays this list in the browser. (See Figure 9.)



Virtual Machine Instances

Instance Id	Current State	AMI Id	DNS Name	Start/Stop	Delete
i-ffe12a94	running	[redacted]	[redacted]	Stop	Delete
i-708a961b	stopping			Start	Delete
i-30a6ba5b	stopped	[redacted]		Start	Delete

Fig. 9. A list of virtual machine instances.

In our current system, once the smart card authenticates the user, it can operate without asking the user PIN again. If the user takes out the card, the application cannot make any more requests to the cloud. The user needs to insert the card and enter the PIN again to continue. The card can also be configured to ask for the PIN for each request.

Performance-wise, using a PSD to sign an API request adds one additional HTTPS request/response round trip between the application client and the server as compared to the application server signing the API request directly. User hardly notice any delay caused by this additional round trip because web applications often rely on a large number of client and server interactions to display a web page.

The signing operation is the HMAC-SHA1 or HMAC-SHA256 hash of the cloud service request. The signing time depends on length of the query, cryptographic algorithm, and communication between the computer and the smart card. Table 1 illustrates an example. The service request is 175 bytes long. The smart card signing time is measured from the computer, which includes the time for sending data to the smart card, computing HMAC-SHA inside the card, and receiving the response.

Table 1. Hashing time examples.

	HMAC-SHA1	HMAC-SHA256
Hash on server	0.31 ms	0.32 ms
Hash on smart card	643 ms	694 ms

Each time entry in Table 1 is an average of 100 hashing times measured from the software running on the computer. The server runs on a Dell Latitude E6410 laptop. Using a smart card to sign is about 600 ms slower. However, this overhead of less than a second is very small compared to the security and usability gain of using a PSD to secure the cloud access. Users can hardly tell any difference because fetching data from the network takes most of the time.

7 Discussions and Future Work

This paper proposes to mitigate key handling risks and complexity by using portable security devices (PSD). The PSD securely exchanges keys with the cloud service provider, stores the keys, and performs operations required by the cloud API using the keys. Neither the application nor the user needs to handle the keys. Hence, this method enhances both security and usability for accessing cloud resources. More specifically, let us look at how the proposed methods solve the problems outlined in Section 1.

1. Key Downloading and Storage. The PSD and the cloud service provider securely establish and exchange keys. The PSD stores the keys in its secure persistent memory. The user does not download and store the keys. This solves the security and the usability problems.
2. Key Usage. The PSD uses the keys stored inside its secure memory to perform required cryptographic operations. The private key never leaves the PSD, and the shared secret key never leaves the PSD or the cloud provider unencrypted. This again solves the associated security and usability problems.
3. Client in Cloud. Even when the client application is running in a cloud, the keys for the application to access other cloud resources are still in PSD. Users do not need to worry about how to load the keys to the application.
4. Key in Memory. Since the PSD performs the cryptographic operations for authenticating cloud API requests, the keys are never in clear text in the computer's memory where the client application runs. Therefore an attacker cannot recover the key from the computer.
5. Server Side Vulnerability. The cloud provider exchanges keys with PSD, and does not present keys to end users. This prevents attackers from stealing keys from the cloud service provider's website.

There are still many unsolved problems that require future work. The PSD is an external device. As such, there is some performance impact to the cloud services. How would this affect usability and how to balance security, usability, and performance are future research subjects. Securely accessing cloud resources from mobile devices is another important future topic.

Currently, cloud providers have their own proprietary ways to authenticate API requests for cloud services. Though challenging, it should be possible to establish a standard protocol with various levels of security. Such a standardized effort can yield tangible benefits for both consumers and service providers. It will not only add security and convenience, but also improve interoperability between different cloud providers.

Acknowledgements

The author would like to thank Kapil Sachdeva, Laurent Castillo, and other Gemalto colleagues for their input and support to this work.

References

1. Amazon Web Services. 2006. The AWS Home Page. [Online] Available from: http://aws. amazon.com/.
2. Amazon Web Services. 2009. AWS Multi-Factor Authentication, [Online] Available from: http://aws.amazon.com/mfa/.
3. Cloud Identity Summit. 2010. [Online] Available from: http://www.cloudidentitysummit.com/.
4. Cloud Security Alliance (CSA). The CSA Home Page. [Online] Available from: www.cloudsecurityalliance.org/.
5. Cloud Security Alliance. 2009. Security Guidance for Critical Areas of Focus in Cloud Computing, v2.1. [Online] Available from: www.cloudsecurityalliance.org/csaguide.pdf.
6. Cloud Security Alliance, 2010. Top Threats to Cloud Computing, v1.0. [Online] Available from: www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.
7. Distributed Management Task Force, Inc., 1992. The DMTF Home Page. [Online] Available from: http://www.dmtf.org/.
8. EMVCo, 2009. The EMVCo Home Page. [Online] Available from: http://www.emvco.com/,
9. EvilPacket.net, 2009. Theft of a Rackspace Cloud API Key. [Online] Available from: http://evilpacket.net/2009/jul/9/theft-rackspace-cloud-api-key/.
10. Garnaat, M. 2009. Managing Your AWS Credentials (Part 2). [Online] Available from: http://www.elastician.com/2009/06/managing-your-aws-credentials-part-2.html.
11. Gemalto. 2006. .NET Card, [Online] Available from: http://www.gemalto.com/ products/dotnet_card/.
12. Jurgensen, T.M. and Guthery, S.B. 2002. Smart Cards – The Developer's Toolkit. Prentice Hall PTR, Upper Saddle River, NJ 07458.
13. Microsoft. 2010. Windows Azure Platform. [Online] Available from: http://www.microsoft.com/windowsazure/.
14. M'Raihi, D. et al, 2005. HOTP: An HMAC-Based One-Time Password Algorithm. IETF RFC 4226. [Online] Available from: http://www.ietf.org/rfc/rfc4226.txt.
15. Open Cloud Computing Interface. 2011. The OCCI Home Page. [Online] Available from: http://occi-wg.org/.
16. Oracle. 2010. Oracle Cloud Resource Model API, version 1.0. [Online] Available from: http://www.oracle.com/technetwork/topics/cloud/oracle-cloud-resource-model-api-154279.pdf.
17. Project Kenai. 2009. The Sun Cloud API. [Online] Available from: http://kenai.com/projects/suncloudapis/pages/Home.
18. Rackspace. 2006. The Rackspace Cloud. [Online] Available from: http://www.rackspacecloud.com/
19. Sachdeva, K., Lu, H.K. and Krishna, K. 2009. A Browser-Based Approach to Smart Card Connectivity. IEEE Workshop on Web 2.0 Security and Privacy, Oakland, California, May 21.
20. SOAtothecloud.com, 2010. Cloud Security Podcast - The question of API Keys. [Online] Available from: http://www.soatothecloud.com/2010/09/cloud-security-podcast-question-of-api.html.
21. Swidler, S. 2009. How to Keep Your AWS Credentials on an EC2 Instance Securely. [Online] Available from: http://shlomoswidler.com/2009/08/how-to-keep-your-aws-credentials-on-ec2.html.
22. Terremark. 2011. Enterprise Cloud Computing from Terremark. [Online] Available from: http://www.terremark.com/services/cloudcomputing.aspx.
23. Vordel. 2009. Cloud Service Broker. [Online] Available from: http://www.vordel.com/products/cloud_service_broker/.
24. Witteman, M. 2002. Advances in Smartcard Security. Information Security Bulletin, July, page 11-22.