Symmetric Key Management Systems

By Arshad Noor

The time has come for the infosec community to address Symmetric Key Management Systems as an application-independent, enterprise-level defense mechanism.

Most security professionals are familiar with symmetric key-based cryptography when presented with terms such as Data Encryption Standard (DES), Triple DES (3DES) and the Advanced Encryption Standard (AES). Some are also familiar with Public Key Infrastructure (PKI) as an enterprise-level solution for managing the life-cycle of digital certificates used with asymmetric-key cryptography. However, the term Symmetric Key Management System (SKMS) – which refers to the discipline of securely generating, escrowing, managing, providing access to, and destroying symmetric encryption keys – will almost always draw blank stares. This is not surprising, because symmetric encryption key management has traditionally been buried in applications performing encryption. These applications primarily focused on business functions, but managed encryption keys as an ancillary function. Consequently, there was no reason to emphasize key management. This article advances the notion that the time has come for the infosec community to address SKMS as an application-independent, enterprise-level defense mechanism that is more effective when addressed separately.

While encryption has been in use for centuries[1], computer-based cryptography entered the general computing field with the advent of the DES algorithm. The primary business uses for this technology was within the military, and later banking. Given the nature of what encryption technology was protecting, implementers were willing to live with custom key-management solutions, however contrived they may have been. With the explosion of the World Wide Web, businesses have been racing to implement business processes on the Internet, bringing sensitive information significantly closer to attacks.

Although businesses have invested billions in firewalls, intrusion detectors, intrusion prevention systems and other defense mechanisms, the US has witnessed more than 300 breach disclosures[2] since the passage of California's Breach Disclosure law[3]. One recent disclosure was from the University of California in Los Angeles (UCLA)[4]. This is the seventh[5] breach disclosure by the University of California across all schools in the UC system, and it reflects a situation completely out of control. Breaches at retailers such as Ralph Lauren, BJ's, DSW and credit card processing companies such as CardSystems Solutions have prompted credit card giants Visa, Mastercard, American Express and Discover to standardize on security requirements for merchants and card-acquirers through the Payment Card Industry's Data Security Standard (PCI-DSS)[6]. One critical element required within PCI-DSS is the encryption of credit card numbers and a robust key management system to accompany encryption.

Rationale

Why is symmetric key management a problem? After all, applications seem to have addressed the problem within the applications for decades, and appear to be continuing to do so. The problem becomes obvious if you are in IT Operations. As an illustration, if you are responsible for managing a point-of-sale (POS) application that accepts

1 History of cryptography. http://en.wikipedia.org/wiki/History_of_cryptography

2 A chronology of data breaches. http://www.privacyrights.org/ar/ChronDataBreaches.htm

3 California's Senate Bill 1386. http://www.strongauth.com/regulations/sb1386/sb1386Index.html

4 Breach at UCLA exposes data on 800,000. http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005925

5 SB1386 Disclosures of Breaches to PII. http://www.strongauth.com/regulations/sb1386/sb-1386Disclosures.html

6 PCI Security Standards Council. https://www.pcisecuritystandards.org/index.htm

credit cards for payment, an e-commerce application that requires credit cards for payment, a payment processing application that communicates with the credit card network for settling transactions, a back-office database that consolidates transactions, and a business analytics application for determining retail fraud, you have five applications that require encryption.

In addition, with the proliferation of laptop and PDA losses and thefts, companies are now mandating encryption on these devices, adding one or two more key-management schemes to the infrastructure. Add database and operating system-specific encryption to the mix, and you round out the picture with at least 8 to 10 key-management infrastructures.

With the proliferation of laptop and PDA losses and thefts, companies are now mandating encryption on these devices.

Since applications are typically purchased from multiple vendors, each vendor, focusing primarily on its own business application, implements encryption and performs key-management functions using its own design. As a result, the IT Operations staff are forced to manage at least 8 to 10 distinct symmetric key-management infrastructures, each with its own technology, training, documentation, procedures and audits. (PCI-DSS regulated entities are required to perform annual audits of any system that stores credit cards.) Not only does this border on the ridiculous, more importantly, it raises total cost of ownership (TCO). One might even argue the potential danger of a vulnerability in the security strategy, because, with so many pots cooking on the stove, something could get burned.

Solution

Presented with the problem in this perspective, the logical solution springs to clarity: the key-management capability needs to be abstracted from the applications that use it. Such a solution is not unlike the Domain Name System (DNS) for hostname-IP address resolution, or a Relational Database Management System (RDBMS) for data management.

In 2006 an open-source software product was released on the Internet that struck at the heart of this problem[7]. Architected along the lines of DNS, the completely free software abstracts symmetric key-management functions from applications and consolidates them on one or more centralized Symmetric Key Services (SKS) servers on the network. Using a client-side API, applications on most platforms can make requests for symmetric key services without knowing the semantics of symmetric key management. Designed to be extremely secure, the SKMS architecture also allows for business continuity in the face of network failures, massive scalability and the use of many well-understood technical standards.

Architecture

An SKMS, as defined within the context of this architecture, consists of at least two centralized SKS servers – a primary and a disaster recovery server – and any number of clients using the Symmetric Key Client Library (SKCL) to request services from the SKS servers. (While they are referred to as clients, the client software may themselves be database servers, web servers, application servers and/or any business application.) The XML-based protocol between the SKCL and the SKS servers, known as the Symmetric Key Services Markup Language (SKSML), has a technical committee (open to anyone) that formed recently at OASIS to consider standardizing this protocol on a royalty-free basis[8].

Each SKS server consists of:

- A server-class computer running an operating system – typically Linux, UNIX or Windows – that has a compliant Java Virtual Machine (JVM) available for it
- A relational database that serves as the storehouse for the symmetric encryption keys
- A J2EE-compliant application server to respond to requests over the network, serving as the workhorse of the SKMS
- A JCE-compliant cryptographic provider to perform the cryptographic operations of key generation, key protection, digital signing, verification, etc.
- An optional, *but strongly recommended*, Hardware Security Module (HSM) or Trusted Platform Module (TPM) for securely storing the cryptographic keys that protect the database's contents
- The SKS server software itself, consisting of an Enterprise Archive (EAR) and a Web archive (WAR) file for the administration console, along with ancillary utilities

Each SKCL client platform consists of:

- A client machine running an operating system – once again, typically Linux, UNIX or Windows, but includes the OS/400 – that has a compliant Java Virtual Machine (JVM) available for it
- A JCE-compliant cryptographic provider to perform the cryptographic operations of encryption, decryption, digital signing, verification, etc.
- An optional, *but highly recommended*, Trusted Platform Module (TPM), smartcard or other USB-based cryptographic token for securely storing the cryptographic keys that protect the clients' authentication credentials
- The SKCL software itself, consisting of an API callable by Java applications for communicating with the SKS server and performing cryptographic functions (non-Java applications have the option of either using a Java Native Interface (JNI) library to call the SKCL, or communicating with the SKS server directly using the SKSML protocol)

The SKSML protocol itself is extremely simple, and consists of:

- A call from the client to request a symmetric key – new or existing – from the SKS server
- A call from the client to request key-caching policy information from the SKS server
- A response from the SKS server containing the symmetric key and the key's use policy
- A response from the SKS server containing the key-caching policy

7 StrongKey. http://www.strongkey.org

8 OASIS Enterprise Key Management Infrastructure Technical Committee. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ekmi

- A fault message from the SKS server, if either of the two calls does not succeed

Security features

Given the sensitivity of the information managed within the SKMS, the architecture is predicated on an extraordinary level of security. (As with any security architecture, the controls and procedures in place at any specific implementation determine the degree of vulnerability the SKMS will have against attacks, so please don't assume these controls are bulletproof and you can skimp on other aspects of security.) The SKMS incorporates the following security features:

- All symmetric keys are generated using any number of compliant cryptographic providers, thus allowing sites to use whatever level of sophistication is desired for their implementation
- All symmetric encryption keys are themselves encrypted using multiple RSA asymmetric keys
- All database records on the SKS server are digitally signed before storage, and verified upon retrieval to ensure their integrity hasn't been compromised
- All administrative operations through the console are digitally signed and maintained in a history log for audit purposes, and verified upon retrieval
- All administrative operations through the console require SSL/TLS-based client authentication
- Only digitally signed client requests are accepted by the SKS server from SKCL clients
- Only digitally signed responses from the SKS server are accepted by SKCL clients
- All symmetric keys are transported, encrypted for the specific client making the request
- All cached keys on the client are digitally signed and encrypted on storage, decrypted and verified upon retrieval to ensure their integrity
- All private keys of the digital certificates can be stored on FIPS-certified cryptographic tokens ranging from software to smartcards, TPMs to HSMs, to ensure their security

Operations

Refer to Figure 1 for the following discussion. When a client – be it a laptop, a DB application or an e-commerce Web server – needs a symmetric key to encrypt some information, it makes a request for a new symmetric key to the linked-in SKCL (or directly to the SKS server if it has implemented the protocol itself).

The SKCL checks its key-cache to determine if it has any cached symmetric keys that are valid for use. If so, it retrieves the key, decrypts it, verifies its integrity, checks its KeyUsePolicy (every symmetric key object has an encryption policy embedded in it, previously defined by the site Security Officer) and then hands the requesting application the symmetric key for use. If the application chooses not to use the SKCL for the actual encryption/decryption operations, it is expected to use the key in conformance with the embedded KeyUsePolicy.

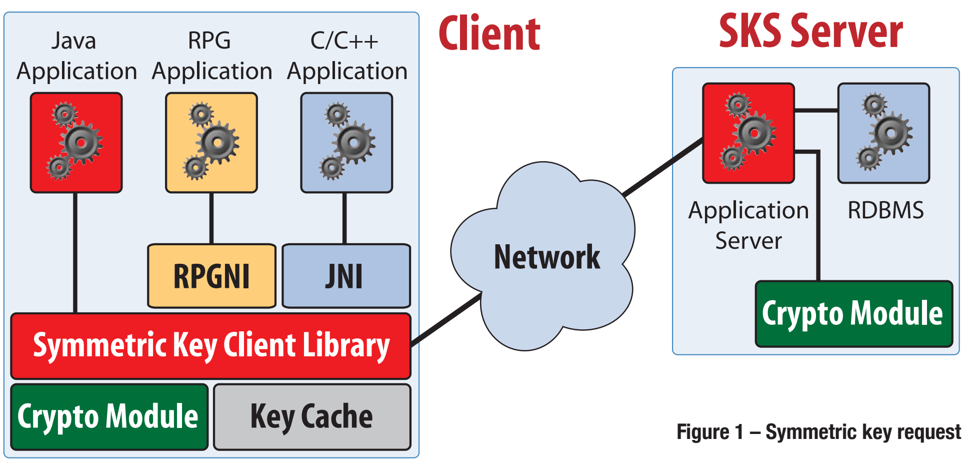

Figure 1 – Symmetric key request

If any of the local checks result in no valid symmetric key being available for use, the SKCL creates a new symmetric-key request, digitally signs it with its authentication credentials, and sends the request to one of its pre-configured SKS servers as an OASIS Web Services Security (WSS)-compliant SOAP request. It is noteworthy to mention here, that since all requests and responses between the SKCL and the SKS servers are secured (digitally signed and encrypted) at the message level, transport-level security (SSL/TLS or IPSec) is not required for the operations of the SKMS; plain old HTTP is sufficient. Administration console communications, however, do rely on mutually authenticated SSL/TLS sessions.

The SKS server, upon receiving such a request, verifies the authenticity and integrity of the request, determines the authorization and the symmetric-key policy in force for the requester (or the default policy), generates a new symmetric key based on this policy, assigns it a Global Key-ID (GKID), escrows the key (which includes encrypting it with multiple RSA keys), encrypts the key with the requester's transport digital certificate, logs the transaction details (which includes digitally signing the transaction) and responds to the client with a WSS-compliant SOAP response.

The SKCL client, upon receiving the response, verifies the authenticity and integrity of the request, caches the secured object if so configured, decrypts the symmetric key and the embedded key-use policy, and returns it to the calling application. The calling application at this time may choose to have the SKCL perform the actual encryption, or perform it itself.

A similar process is repeated when a client application needs to decrypt a previously encrypted object such as a file, directory of files, database record, etc. The application determines the GKID of the

As with any security architecture, the controls and procedures in place at any specific implementation determine the degree of vulnerability the SKMS will have against attacks.

symmetric key it needs (which would have been previously stored with the encrypted ciphertext) and makes a request for this key to the SKCL. The SKCL checks to see if the requested key is in the key-cache. If it is, it goes through the standard security checks and returns the symmetric key to the application; if not, it makes a request to the SKS server for this symmetric key. Upon receiving the request and after the standard security checks, the SKS server responds with the symmetric key to the client. If the key does not exist for any reason, or the client is not authorized to receive the key, or for other error conditions, the SKS server returns a SOAP Fault to the requesting client.

Existing designs – where a single key typically encrypts an entire database or dataset – magnify the loss associated with the loss of that single key.

It is noteworthy to mention that given this operational infrastructure, use of a unique symmetric key to encrypt every record in a database is feasible. With such an encryption policy, the breach of any key reduces the exposure of the database down to just a single record. This is in stark contrast to existing designs, where a single key typically encrypts an entire database or dataset, thus magnifying the loss associated with the loss of that single key.

Implementation

The construction of an SKMS will typically begin with the creation of a PKI – or procurement of PKI services – to manage the issuance of digital certificates to every client. The architecture deliberately eschewed the use of User ID/Password for authentication because of their inability to prevent attacks against single-factor credentials. The clients and servers in an SKMS use digital certificates for authentication, and secure storage and transport of symmetric keys within the infrastructure. (Notwithstanding the use of digital certificates, the administration console allows an Operations or Security officer to "deactivate" any client or server on the SKMS network without revoking the digital certificate of the affected entity.)

Simultaneously, the application that will use the SKCL is modified to integrate the API and accommodate the encrypted data (ciphertext) and the GKID in its database. This raises a valid question of commercial off-the-shelf (COTS) software: How does one use the SKMS if a specific COTS at a site does not support it? Currently we are at a stage of the SKMS' evolution, just as DNS and RDBMS were at their inception. Before the creation of these "abstraction" technologies, applications had to resolve hostname-IP addresses and perform data management on their own. As DNS and RDBMS protocols and APIs became standards, application developers abandoned their proprietary implementations to adopt industry standards – the monetary benefits were too good to ignore. It is anticipated that SKSML will be adopted faster than DNS and the RDBMS, because of the same benefits that would accrue to independent software vendors, and also due to the regulatory and TCO pressures on IT organizations.

Multiple SKS servers are deployed (installation instructions are available at www.strongkey.org), and encryption policies configured on the servers, while digital certificates are issued to clients that will communicate with the servers. The applications are now ready to start requesting key-management services from the SKS servers. The SKMS transitions to Production status at this point, and traditional operational activities take over (backup, configuration management, DR, etc.).

Conclusion

While symmetric encryption has been in use for decades within general computing, we have reached a confluence of inflection points in technology, the Internet and in regulatory affairs, that require IT organizations to implement Symmetric Key Management Systems (SKMS) as independent infrastructures. Using the newly released open-source software, and the soon-to-come Symmetric Key Services Markup Language (SKSML) standard from OASIS, IT organizations have another – and perhaps, one of the most effective – defense weapon in their arsenal against an increasingly hostile Internet.

About the Author

Arshad Noor is the architect and primary developer of the open-source StrongKey Symmetric Key Management software. He is also the Co-Chair of the OASIS Technical Committee on Enterprise Key Management Infrastructures that hopes to standardize the SKSML protocol. He can be reached at arshad.noor@strongauth.com.