In early 2005, NSA announced Suite B, a set of commercial cryptographic building blocks suitable for use in protecting both sensitive but unclassified information as well as classified information. Developers and enterprises should aim to meet these standards, which include recommendations for key agreement, a specific cryptographic function required for most security systems. This article focuses on the two key agreement schemes that are in the Suite B recommendations: the Elliptic Curve Diffie-Hellman (ECDH) and Elliptic Curve Menezes-Qu-Vanstone (ECMQV).

Today, systems architects and design engineers have many cryptographic algorithm and protocol decisions to make when developing secure products for government use. Such considerations begin with the sensitivity and lifetime of the information that will be processed by the proposed system. Additional considerations include the types of security services, such as confidentiality and integrity, that the system must provide.

To provide security services and build trust into an embedded military system, a variety of fundamental cryptographic building blocks (primitives) are required, ranging from symmetric ciphers to hash functions to public key management and digital signature algorithms. The selection of these primitives should be the result of careful analysis. Unfortunately, when these primitives are not explicitly specified by a government source, the typical engineering practice is for designers simply to choose their favorite symmetric cipher, pick a key management system, and produce a secure product that fails to meet the actual information security requirements.

Even when using standardized protocols such as IPSec or TLS, care must be taken to select the appropriate underlying cryptographic primitives in order to configure these protocols to provide the necessary cryptographic strength for the applications. In addition to general cryptographic strength, selection criteria must also consider the capabilities of the underlying platform to perform the cryptographic operations in terms of CPU power, bandwidth, power consumption, and memory requirements; improper selection can adversely affect system performance, cost of goods, and the user experience. All of these are critical design parameters for embedded systems such as manpack radios, satellite systems, and covert communications systems.

This article examines one specific cryptographic function that is required in most security systems: key agreement. Key agreement is a mechanism for two communicating parties to establish a symmetric key in common so that they may encrypt and decrypt information. But before we get into specifics, it’s important to look at the context in today’s security market and the paradigm shift currently underway in public-key cryptography.

New government requirements
In early 2005, the National Security Agency (NSA) announced Suite B, a set of commercial cryptographic primitives suitable for use in protecting both sensitive but unclassified information and classified information. The Suite B primitives are shown in Table 1.

The announcement of Suite B is significant for several reasons:

- RSA, Diffie-Hellman, and DSA are not allowed; only ECC-based public key cryptography is allowed.
- Triple DES is not allowed; only AES is allowed.
- MD5 and SHA-1 are not allowed; only SHA-256 and above are allowed.

Reprinted from MILITARY EMBEDDED SYSTEMS SPRING 2006
This set of primitives raises the bar for what must be considered for use to protect sensitive data. Most commercial systems today use triple DES or AES, SHA-1 and RSA with at most a 2048-bit modulus. (If it were allowed, the RSA Suite B equivalent for use with AES-128 would be 3072 bits.) These combinations are insufficient according to these recommendations by the U.S. government.

From a historical perspective, it is interesting to note that DES was introduced in 1977 for the protection of sensitive but unclassified information. After almost 30 years, Suite B provides a critical update to these aging systems.

The need for key agreement protocols
All secure systems, such as the STU-III, GPS SAASM, IFF, and others, use cryptography to protect the integrity and confidentiality of the data. In its Suite B algorithm selection, NSA chose the symmetric algorithm, Advanced Encryption Standard (AES), because it can provide strong cryptanalytic protection beyond the year 2031. Symmetric ciphers, such as AES, use the same secret key to both encrypt and decrypt data. Therefore, some key management method must be used so that both the sender and receiver can use the same AES key.

While a courier could be used to manually distribute the keying material, this is not practical for a variety of reasons, including:

» Modern networked systems require frequent key changes, often per communications session.
» Certain systems, such as space-based platforms, are inaccessible for manual key distribution.

(continued on page 30)

Suite B primitives

<table>
<thead>
<tr>
<th>Suite B Classification Level</th>
<th>Security Level</th>
<th>Block Ciphers (Minimum)</th>
<th>Hash Algorithms (Minimum)</th>
<th>RSA &amp; Diffie-Hellman Key Sizes</th>
<th>ECC Key Management &amp; Signature Protocols</th>
<th>ECC Field Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitive but Unclassified</td>
<td>128-bit</td>
<td>AES-128</td>
<td>SHA-256</td>
<td>Not Allowed</td>
<td>ECDSA or ECMQV</td>
<td>≥ 256 bits</td>
</tr>
<tr>
<td>Classified</td>
<td>256-bit</td>
<td>AES-256</td>
<td>SHA-384</td>
<td>Not Allowed</td>
<td>ECDSA or ECMQV</td>
<td>≥ 384 bits</td>
</tr>
</tbody>
</table>

Table 1

Glossary

AES (Advanced Encryption Standard) – A NIST-standard symmetric data encryption cipher that uses key sizes of 128, 192, and 256 bits. NIST chose Rijndael as the block cipher for AES. AES provides a very high level of security for the foreseeable future.

ECC (Elliptic Curve Cryptography) – A public-key cryptosystem based on the properties of elliptic curves for efficient encryption and digital signing. The security of ECC is based on the difficulty of the elliptic curve discrete logarithm problem. ECC is the only proven public key technology that scales in a practical way over time.

IFF (Identity Friend or Foe) – Identifier schemes used to distinguish friendly troops and vehicles from those of the enemy.

IPSec (IP Security) – A set of protocols developed by the IETF to support secure exchange of packets inside IP at the network layer. IPSec turns an Internet connection into a private, secure connection. IPSec is used to provide a Virtual Private Network using the public Internet for connectivity. IKE (Internet Key Exchange) is key exchange protocol that is used to set up an IPSec secure connection.

PIV (Personal Identity Verification) – A smart-card-based system used for access control.

RSA – A legacy public-key cryptosystem based on the factoring problem, named after its developers, Rivest, Shamir, and Adleman.

SHA (Secure Hash Algorithm) – The algorithm used in the Digital Signature Standard to produce a hash value. SHA-1 produces a 160-bit hash value but is under cryptographic attack and therefore should not be used for new applications. With the introduction of the AES, which offers three key sizes (128, 192, and 256 bits), there has been a need for a companion hash algorithm with a similar level of security. The newer SHA-256, SHA-284, and SHA-512 hash algorithms, proposed by NIST in 2001, comply with these enhanced requirements.

Suite B – A set of algorithms for protecting classified and sensitive but unclassified information, specified by the National Security Agency under its cryptographic modernization program. The protocols included in Suite B are Elliptic Curve Diffie-Hellman (ECDH) and Elliptic Curve Menezes-Qu-Vanstone (ECMOV) for key transport and agreement, the Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures, the AES for symmetric encryption, and the SHA. It is important to note that the RSA algorithm is not included in Suite B.

TLS (Transport Layer Security) – An IETF standard cryptographic protocol that provides authentication, integrity assurance, and confidentiality for data exchanged by client/server applications.
A lack of scalability exists: The distribution work grows roughly quadratically with the number of independently keyed nodes.

Therefore, it’s essential to support symmetric algorithms with an automated key management scheme. Appropriate schemes may be either key agreement or key transport.

**Key agreement** protocols are those key management schemes in which each party contributes keying material to produce the final secret key; neither party can predetermine the final secret key. This is in contrast to a key transport protocol in which one party unilaterally determines the secret key and sends it to the other party to the protocol. Diffie-Hellman (DH) and its elliptic curve analogs are common examples of key agreement protocols, whereas RSA is an example of a key transport protocol. The secret key that results from the protocol is then used with a symmetric cipher, such as AES, to provide bulk data encryption.

There are a number of different options for key establishment, each one having several desirable security attributes. However, to help developers meet Suite B recommendations, this article focuses on two schemes for key agreement: the Elliptic Curve Diffie-Hellman and Elliptic Curve Menezes-Qu-Vanstone.

**Elliptic Curve Diffie Hellman key agreement protocol**

ECDH is the elliptic curve analog of the traditional Diffie-Hellman key agreement algorithm. The Diffie-Hellman method requires no prior contact between the two parties. Each party generates a public key and a private key and they exchange the public keys. Each party then combines its private key with the other party’s remote key to form the shared secret. This method is also known as carrying out an ECDH key agreement. The ECDH protocol is shown in Figure 1. Here P is a point on a suitably selected elliptic curve.

In the ECDH protocol, Alice and the Bank generate their public and private key pairs (X, x) and (Y, y) respectively. When a communications session is initiated, Alice and the Bank exchange their public numbers. Alice then combines the Bank’s public number, Y, with her private key, x, to create the shared secret \( xyP \). The Bank performs a similar computation. This shared secret is known only to Alice and the Bank; no third party observing the exchanged public numbers can calculate the shared secret.

It is important to note that the shared secret, or a portion of it, must not be used directly as a key for a symmetric cipher. Instead, the symmetric key should be produced from a function that uses all bits of the shared secret and removes any positional bias. Specifically, a Key Derivation Function (KDF), such as that specified in ANSI X9.63, should be used to create the key for the symmetric cipher.

One can see that the main property of the ECDH protocol is that two parties can exchange public information (their public keys) and create a shared secret in common; however, it must be noted that this property alone is insufficient to create a secure key management protocol.

For instance, these protocols are subject to a *man-in-the-middle* attack in which an attacker inserts himself in the communications path between the parties so that Alice and he exchange public numbers and he and the Bank exchange another set of public numbers. This will result in Alice sharing a secret key with the attacker, the attacker and the Bank sharing another secret key, and the attacker being able to read and substitute all data sent between Alice and the Bank.

Therefore, when selecting a key management protocol, one must consider this and other properties such as authentication, forward secrecy, and key confirmation. The more of these properties that can be achieved by a key agreement protocol, the more secure and the more trusted the protocol will be.

For example, ANSI X9.63 defines 13 key agreement protocols involving static keys only, ephemeral keys only, static and ephemeral keys, unidirectional data flow, and bidirectional data flows. Of these, only two of the bidirectional protocols can achieve all the desired security properties: the Station-to-Station Scheme and ECMQV with key confirmation. Of these two schemes, ECMQV is computationally the most efficient. The security and performance reasons are likely why NSA selected ECMQV for inclusion in the Suite B cryptographic primitives.

**Elliptic Curve Menezes-Qu-Vanstone key agreement protocol**

The ECMQV key agreement method is used to establish a shared secret between parties who already possess trusted copies of each other’s public keys. Both parties still generate public and private keys and then exchange public keys. However, upon receipt of the other party’s public key, each party calculates a quantity called an *implicit signature* using their own private key and the other party’s public key. The shared secret is then generated from the implicit signature. The term *implicit signature* is used to indicate that the shared secrets will not agree if the other party’s public key is not employed, thus giving implicit verification that the remote secret was generated by the remote party. An attempt at interception will fail as the shared secrets will not be the same shared secrets because the adversary’s private key is not linked to either trusted public key.

Let’s return to the example of Alice communicating with her Bank. If Alice has the Bank’s public key and the Bank has Alice’s public key, then the ECMQV key exchange may be used.

---

**Figure 1**
Anyone intercepting the transmissions and substituting another remote key is unable to communicate because the resulting shared secrets differ.

ECMQV protocol is shown in Figure 2.

![ECMQV Protocol Diagram](image)

ECMQV: Security and performance

ECMQV primarily owes its efficiency over the Station-to-Station (STS) protocol to the fact that it uses implicit signatures to ensure that the data contributed by the parties is authentic and complete. STS must use a standard-explicit signature, such as ECDSA, which is computationally more expensive. In fact, the dominant calculations in ECMQV are only 1.5 scalar point multiples. As can be seen, the quantity SA acts as a signature on Alice’s ephemeral public key, and only Alice can produce it. The implicit nature of the signature is because the Bank indirectly verifies it when deriving the shared secret since each party’s shared secret will not agree if these signatures are invalid.

Selecting and implementing the optimum scheme

But what does all this mean to implementing key agreement within a security system? How do you choose the right key agreement protocol? A good starting point is whether or not your system must interact with legacy systems using traditional RSA or Diffie-Hellman schemes. If so, you will need to plan for algorithm agility to accommodate these schemes as well as use ECC in order to provide both backward compatibility and Suite B compliance for the future.

For new designs, the Suite B algorithms are the clear choice. Because of its smaller data quantities and efficient computational characteristics, ECC is also a clear choice for devices that are CPU, bandwidth, and/or battery power constrained such as munitions-embedded systems, PIV cards, or IFF devices. Also keep in mind that you must use homogeneous levels of security among your cryptoprimitives: Specifically, AES-128 must be used with at least SHA-256 and ECC functions using a 256-bit field size.

In terms of whether to use ECDH or ECMQV in your system, this selection should be governed by the security properties you need the key agreement scheme to meet, and whether or not you will be using both static and ephemeral keys. ECDH can be used with...
Alice calculates $S_A$ (called the implicit signature):

$$S_A = (x + \chi a) \mod n$$

where $h$ is a co-factor defined in P1363.

The Bank calculates $S_B$ in a similar manner:

$$S_B = (y + \gamma b) \mod n$$

Both Alice and The Bank calculate a shared secret $K$:

$$K = hS_A (Y + \gamma B) = hS_B (X + \chi A).$$

either or both static and ephemeral keys with differing security properties for the various combinations. ECMQV requires the use of both static and ephemeral keys.

As an example of meeting security properties, ECDH, using just ephemeral keys, can only achieve the security property of known key security. This property provides the assurance that the resulting secret key will not be compromised if other secret keys are compromised. On the other hand, ECMQV, using both static and ephemeral keys, can meet the full spectrum of security properties, thus making it the preferred choice for systems demanding complete security. Both ECDH and ECMQV are widely supported in standards, including those for TLS and IKE.

The bottom line is that you need to carefully consider key establishment schemes whenever you are designing a protocol or implementing a security system. As we have shown, there are many different key establishment schemes endorsed by the standards. So your choice comes down to carefully weighing the tradeoffs between the different security attributes and the performance attributes. Generally, key establishment using ECC and ECMQV yield the best results in terms of the security performance tradeoff.

William Lattin is chief technology officer of Certicom Corp. and is responsible for leading Certicom’s technology strategy. During the past nine years, William was the managing director of SecureField, an information security consultancy that specializes in cryptographic product design and network security. He is chair of the Standards for Efficient Cryptography Group (SECG), an industry consortium that develops commercial standards to facilitate the adoption of efficient cryptography and interoperability across a wide range of computing platforms. He holds a BSEE and an MSEE from Stanford University and UC Santa Barbara respectively.

To learn more, contact William at:

Certicom Corp.
393 Vintage Park Drive
Foster City, CA 94404
Tel: 650-655-3950
E-mail: blattin@certicom.com
Website: www.certicom.com