# Project

RSA and ECDH Key transport and key Wrapping

|  |  |
| --- | --- |
| Key agreement | A key establishment procedure where the resultant secret keying material is a function of information contributed by two participants, so that no party can predetermine the value of the secret keying material independently from the contributions of the other party. Contrast with key transport. |
| Key transport | A key establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement. |
| Key wrapping | In this Recommendation, key wrapping is a method of protecting keying material using a symmetric-key-based authenticated encryption method, such as a block cipher key-wrapping mode specified in [SP 800-38F] that provides both confidentiality and integrity protection. |
| Key-wrapping key | In this Recommendation, a key-wrapping key is a symmetric key established through a key-agreement transaction and used with a key-wrapping algorithm to protect the keying material to be transported. |
| Keying material | Data that is represented as a binary string such that any non-overlapping segments of the string with the required lengths can be used as symmetric cryptographic keys. In this Recommendation, keying material is derived from a shared secret established during an execution of a key-establishment scheme or generated by the sender in a key transport scheme. As used in this Recommendation, secret keying material may include keys, secret initialization vectors, and other secret parameters. |

# Project:

1. Use RSA 2048 bit to perform key transport. A large file is encrypted using GCM (AES 256-bit) and the AES key (+ IV) is wrapped and delivered using the RSA key transport.
2. Generate key pairs for Alice and bob and save in PA.txt, SA.txt, PB.txt and SB.txt (Hex)
3. Generate an AES key and IV using controlled seed.

// Generating Time Seed

long startime11 = System.currentTimeMillis();

long nanoGMT2 = System.nanoTime();

long a = new Date().getTime();

ByteBuffer buffer = ByteBuffer.allocate(8);

buffer.putLong(a);

byte [] b = buffer.array();

byte[] nanoBytes = ByteBuffer.allocate(8).putLong(nanoGMT2).array();

ByteArrayOutputStream outputStream = new ByteArrayOutputStream( );

outputStream.write( b );

outputStream.write( nanoBytes );

byte[] TimeSeed = outputStream.toByteArray( );

1. Use AES GCM to encrypt plain.txt (given in this folder and it is a zipped pdf file) and save in cipher.txt (Hex)
2. Use SP 800-38F to wrap the AES key and IV
3. Use SP 800-56B (Figure 3) for key transport from Alice to Bob
4. Alice signs the signature for key transport
5. Bob verifies the signature
6. Bob unwrap the AES key and IV
7. Bob decrypts the ciphert.txt (if you rename plain.txt to plain.zip, you can upzip it to a pdf file)
8. Use ECDH K-571 to perform key transport. A large file is encrypted using GCM (AES 256-bit) and the AES key (+IV) is wrapped and delivered using the ECDH key transport.
9. Generate private keys for Alice and bob using controlled seed save in SA.txt, and SB.txt (Hex).
10. Generate public keys for Alice and bob and save in PA.txt, and PB.txt (Hex)
11. Generate an AES key and IV by hashing the shared secret + a nounce. The shared secret is produced by ECDH (SP 800-56C).
12. Use AES GCM to encrypt plain.txt (given in this folder) and save in cipher.txt (Hex)
13. Use SP 800-56A ECDH for key transport from Alice to Bob
    * Generate Alice’s ephemeral key pair
    * Use Bob’s static public key and Alice’s ephemeral private key to form an shared secret
    * Use SP 800-56C to generate the AES key and IV
14. Use SP 800-38F to wrap the AES key and IV + Alice’s public ephemeral key
15. Alice signs the signature (ECDSA) for key transport
16. Bob verifies the signature
17. Bob unwrap the AES key and IV
18. Bob decrypts the ciphert.txt

(Blue text means use existing result)

1. Use password to generate keying material using the strongest method in SP 800-132. Consider how to protect salt in a key transport. The salt can be wrapped.
2. Generate a 512 bit salt using a controlled seed. (save in saltt.txt)
3. Generate a AES key and IV using SP 800-132 (save in keyIV.txt)
4. Use AES GCM to encrypt plain.txt (given in this folder) and save in cipher.txt (Hex)
5. Use SP 800-38F to wrap the Salt. (save in Saltwrap.txt)
6. Upwarap saltt.txt
7. Regenerate a AES key and IV
8. Decrypt the ciphert.txt

# Final:

1. Discuss the security strength for Project part 1 using a table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RSA | AES key | IV | Key wrap key | …. | Total |
| 1024 | 128 |  |  |  |  |
| 256 |  |  |  |  |
| 2048 | 128 |  |  |  |  |
| 256 |  |  |  |  |
| 4096 | 128 |  |  |  |  |
| 256 |  |  |  |  |

1. Discuss the security strength for Project part 2 using a table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ECDH | Private key length | ….. | AES | IV | Key wrap key | …. | Total |
| P-256 |  |  | 128 |  |  |  |  |
|  |  | 256 |  |  |  |  |
| P-384 |  |  | 128 |  |  |  |  |
|  |  | 256 |  |  |  |  |
| P-521 |  |  | 128 |  |  |  |  |
|  |  | 256 |  |  |  |  |
| K-571 |  |  | 128 |  |  |  |  |
|  |  | 256 |  |  |  |  |

1. Discuss the security strength for Project part 3 using a table. Create a table to include all input parameters.
2. Compare the ECDSA and RSA-DSA speed for signing plain.txt for the same strength.
3. Compare ECDH and DH speed for generating a shared secret.
4. (bonus) Password cracking against Windows 7.

Tools:

* John the Ripper password cracker by Openwall Project

For Unix, Mac OS X, Windows, Linux, freeware

* Offline NT Password & Registry Editor by Petter Nordahl-Hagen

Windows NT/2000/XP/2003/Vista/7 (32- and 64-bit), freeware

* MDcrack by Gregory Duchemin

Unix / Win32, freeware

* RainbowCrack

For Windows, Linux, freeware

* Hash Suite by Alain Espinosa

Windows XP/2003/Vista/7, shareware, free trial, $29.95

* Proactive System Password Recovery by ElcomSoft

Windows, $79

# Help

## NIST Special Publication 800-38F Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping

key wrapping, i.e., the protection of the confidentiality and integrity of cryptographic keys. In addition to describing existing methods, this publication specifies two new, deterministic authenticated encryption modes of operation of the Advanced Encryption Standard (AES) algorithm: the AES Key Wrap (KW) mode and the AES Key Wrap With Padding (KWP) mode.

## Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography

**NIST Special Publication 800-56A**

**August 2012**

**DLC-Based Key Transport**

A DLC-based key-transport scheme uses both a key-agreement scheme and a key-wrapping algorithm in a single transaction to establish keying material. During this transaction, a key-wrapping key **shall** be established using an **approved** key-agreement scheme. This key **shall** be used by party U to wrap secret keying material using an **approved** key-wrapping algorithm; the wrapped keying material is then sent to party V (i.e., party U in the key-agreement scheme will be the key-transport sender, and party V will be the key-transport receiver). **Approved** key-wrapping algorithms are provided in SP 800-38F.

To comply with this Recommendation, the key-transport transaction **shall** use only **approved** key-agreement schemes that employ party V’s static key pair8 and require an ephemeral contribution by party U9. In particular, a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme **shall** be used in which party U is the intended key-transport sender; a C(2e, 0s) scheme **shall not** be used to establish the key-wrapping key (regardless of which party is the intended key-transport sender). Although other methods may be used by protocols that incorporate key transport, this Recommendation makes no statement as to the adequacy of those methods.

Key confirmation may optionally be provided by party V following the unwrapping of the received keying material, either instead of or in addition to any key confirmation that may be performed as part of the key-agreement scheme. When key confirmation is performed following the unwrapping process in accordance with this Recommendation, parties U and V **shall** have agreed upon an **approved** MAC algorithm and associated parameters (see Table 8 and Figure 9 in Section 5.9.3).

**7.1**

**Key Transport Scheme**

The DLC-based key-transport scheme is as follows:

1. An agreed-upon C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme is used between party U and party V to establish *DerivedKeyingMaterial*, which includes a *KeyWrappingKey* that will subsequently be used by party U for key-transport. Key confirmation (as specified in Section 5.9 and Section 6) may optionally be incorporated in the key-agreement scheme to provide assurance that the *DerivedKeyingMaterial* is the same for both parties.
2. Party U obtains the *KeyWrappingKey* from the *DerivedKeyingMaterial*.
3. Party U selects secret keying material, *TransportedKeyingMaterial*, to transport to the receiver. If key confirmation is to be performed following key-transport, this *TransportedKeyingMaterial* **shall** include a fresh (i.e., new) *MacKey* to be used for key confirmation and the *KeyData* to be used subsequent to key transport (see Section 7.2).
4. Party U calculates *WrappedKeyingMaterial* = KeyWrap(*KeyWrappingKey, TransportedKeyingMaterial*) using KeyWrap( ), an **approved** key wrapping algorithm.
5. Party U sends *WrappedKeyingMaterial* to party V.
6. Party V receives *WrappedKeyingMaterial* from party U.
7. Party V obtains the *KeyWrappingKey* from the *DerivedKeyingMaterial*.
8. Party V calculates *TransportedKeyingMaterial* = KeyUnwrap(*KeyWrappingKey, WrappedKeyingMaterial*) using KeyUnwrap( ), the key-unwrapping algorithm that corresponds to KeyWrap( ).
9. If key confirmation is to be performed subsequent to key transport to provide assurance to party U that the correct *TransportedKeyingMaterial* has been obtained by party V, then both parties U and V **shall** proceed as specified in Section 7.2.

Note that if the key-agreement scheme used in Step 1 is such that party V does not contribute an ephemeral key pair to the calculation of the shared secret (that is, a C(1e, 2s), C(1e, 1s), or C(0e, 2s) scheme has been used) and key confirmation is not included in the key-agreement scheme, then Steps 1 through 5 can be performed by party U without direct involvement of party V. This can be useful in a store-and-forward environment, such as e-mail.

Key transport schemes can be used in broadcast scenarios. In a broadcast scenario, an exception is made to the rule in this Recommendation that ephemeral keys **shall not** be reused (see Section 5.6.3.3). That is, party U may use the same ephemeral key pair in step 1 above in multiple instances of DLC-based key-agreement (employing the same scheme) if the same secret keying material is being transported to multiple entities for use following key transport10, and if all these instances of key transport occur “simultaneously” (or within a short period of time). However, the security properties of the key-establishment scheme may be affected by reusing the ephemeral key in this manner.

**7.2 Key Confirmation for Transported Keying Material**

If key confirmation is to be provided in compliance with this Recommendation following the transport of keying material (as specified in Section 7.1), party U **shall** generate a fresh *MacKey* and include it as part of the *TransportedKeyingMaterial* to be transported (see Section 7.1). The transported *MacKey* **shall** be used for the computation and verification of the *MacTag* provided by party V to party U.

For each instance of key confirmation following key transport, this *MacKey* **shall** be generated anew using an **approved** random bit generator that is instantiated at or above the security strength required for the key establishment transaction. In broadcast scenarios, a different *MacKey* **shall** be included in the *TransportedKeyingMaterial* for each key-transport receiver that is expected to provide key confirmation to party U.

The domain parameter set used by the key-agreement scheme employed to establish the key-wrapping key **shall** be used to determine the minimum *MacKey* length and the length of the *MacTag*, as specified in Tables 8 and 9 in Section 5.9.3.

The transported keying material **shall** be formatted as follows:

*TranportedKeyingMaterial* = *MacKey* || *KeyData*.

The *KeyData* may be null, or may contain keying material to be used subsequent to key transport. The *MacKey* **shall** be used during key confirmation and then immediately destroyed by both party U and party V.

The *MacKey* portion of the transported keying material and an **approved** MAC algorithm (see Sections 5.2 and 5.9.3) are used by each party to compute a *MacTag* (of an appropriate, agreed-upon length) on the *MacData* (see Section 5.9.1.1) represented as

*MacData* = “KC\_KT” || *IDV* || *IDU* || *EphemDataV* || *EphemDataU* || *WrappedKeyingMaterial* { || *Text*},

where *IDV* is the identifier associated with party V, and *IDU* is the identifier associated with party U. These identifiers **shall** be the same as those used to label parties U and V during the key-agreement portion of the key-transport transaction. *EphemDataV* is the ephemeral public key or nonce contributed by party V during the establishment of the key-wrapping key used for key transport; if no ephemeral data was contributed by party V, then *Null* **shall** be used. *EphemDataU* is the ephemeral public key or nonce that was contributed by party U during the establishment of the key-wrapping key. *WrappedKeyingMaterial* is the ciphertext of the keying material that has been transported, and *Text* is an optional bit string that may be used during key-confirmation that is known by both parties.

Party V (the *MacTag* sender) computes a *MacTag* (using the *MacKey* obtained from the *TranportedKeyingMaterial* and *MacData* formed as described above) and provides it to Party U. Party U (the *MacTag* receiver) computes a *MacTag* (using the *MacKey* that was included in the *TranportedKeyingMaterial* and the *MacData* formed as described above)*.* Party U then verifies that this newly computed *MacTag* matches the *MacTag* value provided by party V

## NIST Special Publication 800-57 Recommendation for Key Management – Part 1: General (Revision 3)

[SP800-56A] specifies key-establishment schemes that use discrete-logarithm-based public-key

algorithms. With the key-establishment schemes specified in [SP800-56A], a party may own an

ephemeral key, a static key, or both an ephemeral and a static key. The ephemeral key is used to

provide a new secret for each key-establishment transaction, while the static key (if used in a

PKI with public-key certificates) provides for the authentication of the owner. [SP800-56A]

characterizes each scheme into a class, depending upon how many ephemeral and static keys are

used. Each scheme class has its corresponding security properties.

**RSA Key Establishment**

RSA key-establishment schemes based on the integer-factorization problem have been **approved**

in [SP800-56B]. Four scheme families are specified, two families for key agreement and two for

key transport. Each scheme family has a basic scheme and one or more key confirmation

schemes.

## NIST Special Publication 800-56B Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography

**August 2009**

RSA key transport

## NIST Special Publication 800-132 Recommendation for Password-Based Key Derivation Part 1: Storage Applications

This Recommendation specifies techniques for the derivation of master keys from passwords or passphrases to protect stored electronic data or data protection keys.