Comments Received on the Recommendation for Draft SP 800-56
Key Establishment Schemes

Date: Thu, 30 Jan 2003 12:50:36 -0500 (EST)
From: Hugo Krawczyk <hugo@ee.technion.ac.il>
Message-ID: <Pine.LNX.4.33.0301301207570.18027-100000@e54.watson.ibm.com>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII

I was browsing over your draft document on key establishment techniques (draft 2 - Jan 2003). I am curious to know why the recommendations leave outside some of the most common and practiced methods of key establishment, in particular, several of the main techniques for authenticated Diffie-Hellman exchange. Specifically, there is no method for authenticating an ephemeral DH exchange with a digital signature(such as those used in the STS protocol and the IKE protocol of the IETF and SSH), or those using public-key encryption for authentication (e.g SSL, IKE, SKEME).

I'd appreciate your feedback on this.

Hugo Krawczyk

Response: A conscious decision was made to limit the number of techniques (at least for this pass), since we needed to worry about interoperability, and also cannot fully analyze all techniques available. However, we have looked at many of the techniques that are used in the known protocols and hope that we have included a technique that "matches" many of those that are used. Since the people on our working group are familiar with the ANSI key establishment techniques, and many have participated in their development, their inclusion in our document was a good starting point. The inclusion of our schemes in a protocol that adds additional security attributes, e.g., uses digital signatures, is not discouraged.

NOTES ON NIST SPECIAL PUB 800-56

(Nechvatal, 2/11/03)

1. Sec. 5.1.1.1, paragraph 2: the reference to "the security equivalence table in ... [8]" is obscure. Location of the referenced table should be specified.

2. Sec. 5.1.1.1, last sentence: the phrase "transitive trust" should be explained, or a reference provided.

3. Table 2, Sec. 5.1.1.2: Since there is no variation in the maximum value of ECC cofactor, there is no point in listing it 5 times.

4. Sec. 5.1.2, #2.a: "FIPS 186-3 (revision of FIPS 186-2)" is spelled out, as opposed to "[3]" on the previous page. This is inconsistent and redundant.

5. Sec. 5.2.2.2, #3: "receivesd".

6. In Sec. 8.3.4, p. 59, Sec. 8.3.7, p. 63, and Sec. 8.3.8, p. 64, there are footnotes substantially identical to the footnote on p. 58. These all look like add-ons after the fact. Compression is in order. Better yet, "party has actively participated" should be explained in the regular text when first used on p. 58, and footnotes 3 - 6 should be stricken.NOTES ON NIST SPECIAL PUB 800-56

(Nechvatal, 2/11/03)

1. Sec. 5.1.1.1, paragraph 2: the reference to "the security equivalence table in ... [8]" is obscure. Location of the referenced table should be specified.

2. Sec. 5.1.1.1, last sentence: the phrase "transitive trust" should be explained, or a reference provided.

3. Table 2, Sec. 5.1.1.2: Since there is no variation in the maximum value of ECC cofactor, there is no point in listing it 5 times.

4. Sec. 5.1.2, #2.a: "FIPS 186-3 (revision of FIPS 186-2)" is spelled out, as opposed to "[3]" on the previous page. This is inconsistent and redundant.

5. Sec. 5.2.2.2, #3: "received".

6. In Sec. 8.3.4, p. 59, Sec. 8.3.7, p. 63, and Sec. 8.3.8, p. 64, there are footnotes substantially identical to the footnote on p. 58. These all look like add-ons after the fact. Compression is in order. Better yet, "party

has actively participated" should be explained in the regular text when first used on p. 58, and footnotes 3 - 6 should be stricken.

From: Stephen Kent <kent@bbn.com>
Subject: NIST key management guidelines & IKE v2

The question I have is whether NIST would accept IKE implementations as COMPLIANT with the Key Establishment recommendations or not. If the NIST labs don't hold implementations to a particular ordering of fields I think we are ok. Otherwise we have a problem when a big company creates an IKE compliant implementation but can't get it NIST approved because the fields are in the wrong order.

I would like to see the NIST standards accept IKE implementations as COMPLIANT. That can be done easily by saying order doesn't matter or explicitly by changing the NIST recommendations to allow for alternateorderings of fields. There are so many options already this would seem feasible.

AND

We had this issue with RSA. The FIPS said that it had to do padding according to an ISO standard, but all of the vendors use PKCS#1. I am unaware of any implementations that actually met the NIST standard. NIST eventually came around.

In my opinion, NIST MUST accommodate the IPsec way of doing Diffie-Hellman. At this point, there is no way that the IPsec community is going to change, and NIST must accommodate the IPsec community (unless there is a flaw).

Date: Thu, 27 Mar 2003 15:37:59 -0500
From: Russ Housley <housley@vigilsec.com>
Subject: Diffie-Hellman Schemes
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"; format=flowed

I think that it is important to be able to get FIPS 140 validation of IPsec and TLS implementations; however the current structure of this document does not allow this to be done. The current document only supports the X9.42 construction for Diffie-Hellman. This is not the construction used in IPsec or TLS. I suggest that the schemes be expanded to support IPsec and TLS in addition to X9.42.

Russ

Date: Sat, 29 Mar 2003 11:23:04 -0500
From: David Halasz <dhala@cisco.com>
Subject: NIST SP 800-56 comment

Sent on behalf of Stuart Kerry, Working Group Chair of IEEE 802.11.

IEEE 802.11 would like NIST to allow HMAC-SHA1 and AES-CBC-MAC to be used in approved KDFs (Key Derivation Functions) as alternatives to a one-way hash function in Clause 5.3 of the draft SP 800-56.

A strong theoretical basis exists for using HMAC-SHA1 and AES-CBC-MAC as pseudo-random functions. Also, HMAC-SHA1 and block ciphers in CBC-MAC mode are used widely as pseudo-random functions. Since the heart of any key derivation function is typically a pseudo-random function, it seems appropriate to consider these for use in NIST-approved key derivation functions. The IEEE 802.11 Task Group i has already adopted HMAC-SHA1 and is considering using AES-CBC-MAC.

Dave Halasz
IEEE 802.11i Task Group Chair

Dave Halasz
Cisco Systems, Inc.
320 Springside Drive
Akron, OH 44333

Entrust Inc. Comments on "NIST Special Publication 800-56: Recommendation on Key Establishment Schemes – Draft 2.0"

April 3, 2003

We would like to thank NIST for the thought and effort that went into this document and acknowledge that it is a valuable contribution in achieving an improved practise of security. It is in that spirit that we offer these comments.

1. General

We question the inclusion of such a large number of schemes in this document. It will make interoperability, validation testing and scheme selection by implementers much more difficult than it otherwise would be. From Entrust's perspective as an implementer the only key agreement methods that are currently required are the dhOneFlow for store-and-forward environments and the dhEphem for session-based environments which is usually embedded in a protocol that provides authentication (and the ECC alternatives: One-Pass Diffie Hellman and Ephemeral Unified Model). Possibly the dhStatic (and the ECC alternative: Static Unified Model) could be useful in some environments. However, beyond that we see little advantage in including this large number of options. If it is determined that inclusion of all of these schemes is warranted, then at least including a list of preferred schemes would be helpful.

2. Section 4

The Key Management Guidelines document (NIST SP 800-57) specifically defines *key wrapping* as being only when a key is encrypted using a symmetric key. However, throughout this document the term is used to refer to encrypting a key with either a symmetric key or an asymmetric key. The terminology should be harmonized between these two documents.

3. Section 5.1.1.1

It is not exactly clear from the text if the bit lengths provided in Table 1 are the only bit lengths allowed. If this is the case it should be made clearer.

If only the key sizes in Table 1 are allowed (and a similar table is produced for RSA) we question the choice of key sizes. We appreciate the desire to have a small number of supported key sizes for interoperability and validation testing and to provide key sizes equivalent to all of the AES key sizes. However, we must also prepare for unexpected results that gradually increase the asymmetric key length. If implementations only support 1024, 2048, 3072 and 8192 bit keys, for example a new result that would place 80 bits of security equivalent to 4000 bits of asymmetric key would then require everyone to use 8192 bit asymmetric keys since no other key lengths are supported. This would be less than ideal. Thus, we recommend supporting intermediate key lengths. In particular, 4096 is already somewhat widely deployed and definitely should be supported. It probably also makes sense to include another intermediate length at 6144 bits. Including these key lengths would allow a gradual transition to longer key lengths if that were

required. However, providing further granularity above 8192 bits seems counter productive since advances that would require the use of key sizes of this length would likely also require a complete rethinking of the asymmetric algorithms being used.

We also note that the European NESSIE project recommends using RSA keys of 1536 bits. Does NIST also wish to support this key length?

4. Section 5.1.1.2

It is not clear in this section if only the NIST recommended elliptic curves are allowed. We believe that the NIST curves should be recommended to implement, in order to promote interoperability, but that other curves still are allowed.

5. Section 5.1.2

Another method to receive assurance from a trusted third party about the validity of a set of domain parameters is for them to be hardcoded in the cryptographic module itself. This may be mentioned as a way to obtain the assurance listed in point 3.

6. Section 5.2.2.1

This section states that the owner of a public key shall know what method of public key validation was used. (Similar statements are made in subsequent sections.) Does this mean that the human owner of the key must know this information? This seems rather user-unfriendly. However, there may be an owner's representative or system administrator, for example, which could make the decision for the actual owner/user. If it means that the owner's software must know, it's not clear what this is saying since it is the software that must provide the validation, so it clearly must know.

7. Section 5.2.2.2 and Section 5.2.2.4

We suggest that the public key validation method in ANSI X9.63 Section 5.2.2, Option 4 also be allowed at least in some cases. This method states that the recipient trusts the authenticated owner of the public key for the lifetime of the key that it is being combined with. Otherwise, costly full public key validation will be performed even when it is not required. For example, when ephemeral Diffie-Hellmann is embedded in a Station-to-Station type protocol, then this type of validation provides sufficient security (since the only key possibly being attacked is ephemeral and the only possible attacker is the other legitimate party) and would eliminate the requirement to use full public key validation, which may affect the efficiency of the protocol.

Since co-factor DH is mandated for ECC schemes it would seem reasonable to allow partial public key validation for static public keys as well as ephemeral keys. If partial public key validation were used in this situation the problem wouldn't be noticed until the DH operation was attempted, which is less efficient than detecting it earlier using full public key validation, but this should be an engineering decision left up to implementers.

8. Section 5.2.2.5

The first sentence of this section appears to be partially missing.

9. Sections 5.2.2.5, 5.2.2.6, and 5.2.2.7

It should be made clear that the PKV routines do not necessarily require doing these calculations as specified if they can be done in some other way. For example, the order check might not explicitly be required if the cofactor is 1 and the candidate public key is known to be a non-identity point on the curve. Certainly doing the actual calculation is one way to verify it, but they may be others, at least in some cases.

10. Section 5.2.3

We would like to suggest that IETF RFC 2510 be referenced as providing accepted methods for providing POP in a PKI environment.

11. Section 5.3.1

This document requires identifiers for both the sender and recipient to be included in the key derivation function, and the MAC calculation if key confirmation is provided. However, in some circumstances there is no identifier. In particular in some store and forward environments the sender remains anonymous. This is often a feature of these schemes. Thus, an ability to include a standard "anonymous" identifier should be included or the identifier should be removed when none exists. If NIST does intend to not allow "anonymous" or absent identifiers then this possibility should be explicitly forbidden.

12. Section 5.3.2

A pointer to the actual ASN.1, in X9.42, used to calculate the key derivation function would be useful.

In Input 3.1.2 the value 00000001_{16} does not have the subscript.

We would like to comment on the fact that the choices made in this section make this key derivation function incompatible with the one that is currently used in S/MIME which will make implementing conformant solutions much more difficult.

13. Section 6

It is probably worth noting in this section that the security attributes provided by any particular scheme also depend upon the protocol in which the scheme is embedded. For example, if digital signatures are applied to ephemeral keys that additional security attributes may be provided.

14. Section 6.1.2.3

It should be mentioned in the third paragraph that one common method of authenticating the source of the ephemeral keys is for each party to sign their ephemeral key.

15. Section 6.2.2.3

The last sentence in the third paragraph should be changed from "… would provide this assurance to any party …" to "… would provide this assurance **to derived keying material** to any party …".

The first sentence of the fourth paragraph should remove the phrase “or shared secrets”.

16. Section 6.3.3

The last sentence of the second paragraph should be changed from “Variability can be provided …” to “Variability **of the derived keying material** can be provided …”.

17. Section 7.3

We would like to strongly recommend that NIST include those IFC-based key transport schemes that are in common practice. In particular at least one of PKCS #1 v1.5 and OAEP should be included. Since there are already two schemes in common defining additional schemes will only make interoperability more difficult and hinder availability of conformant products.

18. Section 8

It is not clear if the key confirmation methods described in this section are required when performing key confirmation, or if other methods are also allowed. We recommend that other methods be allowed since the method described in this document is currently not implemented in any products, to our knowledge. In addition, this document seems to mandate certain message flows for providing key confirmation that may, or may not, fit into the protocols required to provide certain services or to obtain the desired security attributes. Is this the intent? Given that NIST is not describing the protocols used, it seems that mandating the message flows could be counterproductive.

Key confirmation for ephemeral keys is commonly performed in protocols such as SSL/TLS and IPSEC. Thus, the statement that key confirmation cannot be performed for these keys is perplexing. Performing key confirmation for ephemeral, unauthenticated keys provides the recipient with the assurance that the other, anonymous party in the key exchange does, in fact have the established key. If the ephemeral key is authenticated it also provides the assurance that the key owner is alive and participating in the key establishment at that time. Is NIST stating that key confirmation for these keys SHALL NOT be performed, or simply that this specification does not support it? If the latter, it should be clarified that there is no technical reason that key confirmation for ephemeral keys cannot be performed and that it may be provided, outside of these recommendations.

The footnotes in this section talk about having a certain number of unpredictable bits in the nonce. For the uninformed reader this may be misinterpreted. For example, if only the strings 111…111 and 000…000 are allowed then none of the bits are predictable, but there is only one bit of entropy. Thus, the footnotes should instead discuss the number of bits of entropy.

19. Section 8.1

As we previously noted, identifiers for both parties are not always available and thus a method should be provided to allow anonymous identifiers or to not include them.

Editorial Comments

Table of Contents section 5.8.2.2.2 – this section is darker than the others and the

indentation is strange for the 5 level entries.

Table 1 is missing a bottom line.

Figure 1 has different lines (codes) for each of the message flow lines. This is true for some other figures in the document. The line codes should be explained somewhere and used consistently.

Figure 2 is missing the bottom line.

Appendix B: References 10 and 18 - The dates of December 2003 must be typos; probably what is meant is December 2002.

From: "Kaliski, Burt" <BKaliski@rsasecurity.com>
Date: Thu, 3 Apr 2003 14:53:21 -0500

RSA Security Comments on NIST Special Publication 800-56, "Recommendation on Key Establishment Schemes", January 2003 draft

Burt Kaliski
bkaliski@rsasecurity.com

April 3, 2003

General comments

In general I found the draft Recommendation to be an excellent document, clearly reflecting considerable thought. Many standards for key establishment have been developed over the years, but often without a clear "system". The draft Recommendation provides that system by several "key" contributions:

* The introduction of "classes" of related schemes based on the number of ephemeral and static key pairs, such that the schemes all have the same "flow" and structure. This helps to "abstract" the scheme so that it can fit particular application requirements. An application can choose a _class_ based on protocol requirements (number of passes, etc.) and security requirements (forward secrecy, responder authentication); and then choose a scheme within a class based on performance and other tradeoffs.

* The specification of key confirmation as a separate service that can be added to any of the schemes. Rather than each standard or scheme "reinventing the wheel" (potentially incorrectly) as regards key confirmation, there is now one common framework that all the schemes can follow.

* The formulation of key transport as a combination of key agreement and symmetric key wrapping. This strategy, also followed in the current draft of ANS X9.44, has the notable advantage that keying material to be transported is handled only by a symmetric key wrapping scheme, regardless of the type of cryptography (discrete logarithm, integer factorization, symmetric) employed by the receiver. The only difference is how the key-wrapping key is obtained. This formulation will simplify implementation as well as testing.

Below, I give several technical comments and also some editorial suggestions. One other issue I would like to raise is patent assurances. NIST has been very attentive to patent issues in other standards efforts it has led, such AES. I am not clear on NIST's intentions for the methods in this draft Recommendation, but would look to NIST for guidance as to which parties may have asserted intellectual property rights on various recommended methods, and that methods are available, where possible, that are free from any encumbrances.

Primary Technical Comments

1. Key validation

I agree with the general intent of the draft Recommendation in requiring parties to have assurance of the validity of public keys and domain parameters, as opposed to requiring parties explicitly to check the mathematical properties of public keys (though this is one method of obtaining such assurance). Many security attributes are conveyed implicitly, including perhaps even more important ones such as the key owner's identity. An implicit form of assurance is thus both appopriate and practical.

Public-key validation is a complex subject because parties may have different objectives depending on the situation. A recipient, in general, wants the owner's public key to be valid so that a key established using

the public key is not compromised. If the owner's public key is combined directly with the recipient's static private key, the recipient wants the owner's public key to be valid for a second, and perhaps more important reason: to avoid compromise of the recipient's private key. This is a consideration in some but not all key establishment schemes (and in particular only in certain schemes in the discrete logarithm family, not in the integer factorization family).

An owner, in general, wants its own public key to be valid for reasons similar to the recipient's first reason: so that an established key is not compromised. However, the owner, if dishonest, may seek to employ an invalid public key in order to obtain information about the recipient's private key (in the situation mentioned above) or the established key (in certain digital rights management applications).

In light of these different situations, the blanket approach in the draft Recommendation where assurance of validity is required for _all_ public keys, whether static or ephemeral, and regardless of the key establishment scheme, may be slightly more than is needed. While I recognize the benefits of having a "safe default", I also want to be sure not to impose requirements on implementations that do not actually add security (and in fact may introduce a false sense of security).

One situation that comes to mind is ephemeral-ephemeral Diffie-Hellman (the dhEphem scheme in 6.1.2.1 and the Ephemeral Unified Model scheme in 6.1.2.2). Here, an attacker can readily substitute _valid_ public keys in an attempt to control the communication (since none of the parties is authenticated). Although public-key validation may help in some situations (in particular, when the attacker is only "in the middle" when the public keys are exchanged, and not during key confirmation or subsequent communication), assurance of public key validity may not offer additional protection in others. That is, it may still be possible to achieve the desired security attributes even without assurance of public key validity, because it has already been assumed that there is no "man-in-the-middle". (Indeed, perhaps the "man-in-the-middle" is detected by other means, e.g., exchange of identifiers by between the users.) This is consistent with the treatment in 6.1.2.3, which states "Despite the fact that these schemes offer very few assurances, they are useful in many applications."

A related observation is that the assurance requirements may be different depending on the key establishment scheme. In general, the level of assurance of public key validity should be at least as high as the value of the established key. But in the schemes where the recipient's private key is combined with the owner's public key, the level should be as high as the value of the recipient's private key. This is likely to be much higher than the value of any individual established key. Accordingly, while the schemes where the recipient's private key _is_ combined with the owner's public key may need a high level of assurance (perhaps Recipient or TTP Full Validation), the schemes where the recipient's private key is _not_ so combined may be just as "safe" with a more modest level of assurance ((say, just TTP Assurance of Implementation Validity, as described below).

Parties of course need more than just the assurance that one another's public keys are valid; they also need the assurance that one another's implementations are correct. Even if my public key is valid, your data may still be at risk if my implementation has errors. This is the primary motivation for implementation validation, which is appropriately required in Clause 10. But if implementation validation is required, there should be some method for a party to obtain assurance that another party's implementation has been validated. Assuming this is provided, i.e., you have assurance that my implementation is valid, then you also have assurance that my public key is valid (assuming that you know that my public key was generated by my implementation). This leads to an additional method of assurance that I believe also should be allowed in 5.2.2.1 and 5.2.2.2: TTP Assurance of Implementation Validity. This method, which can be provided to both the recipient and the owner, may be described as follows:

- TTP Assurance of Implementation Validity -- The recipient [or owner] receives assurance that a trusted third party (trusted by the recipient [or owner]) has validated the key generation routine that generated the public/private key pair.

This method provides not only the assurance that the key generation routine is valid, but also that the key generation routine generated the key pair in question. Some examples of how this assurance might be provided in practice include:

* The owner generates the key pair on a smart card. The owner physically presents the smart card to the recipient. The recipient checks the smart card for a marking indicating that the smart card has been appropriately validated, and then uploads the public key from the smart card.

* The owner does the same in the presence of a certificate authority. The certificate authority issues a certificate for the uploaded public key. The recipient obtains the assurance indirectly via the certificate.

* The owner has a cryptographic module, which has its own public/private key pair. The module's public key has a certificate, issued by the testing laboratory when the module was validated. The module generates the owner's public/private key pair and signs the owner's public key with the module's private key. The owner presents the module's certificate and the signature on the owner's public key to a certificate authority as part of a registration protocol. The certificate authority checks the certificate and the signature, and if they are valid, issues its own certificate for the owner's public key.

TTP Assurance of Implementation Validity may also be provided to the owner. In fact, this method is the "de facto" approach unless the owner validates the implementation itself. The Owner Generation method, for instance, presumes that the implementation is valid. The owner is relying on the manufacturer or on a testing laboratory for the assurance that the implementation is correct.

Another method of assurance I would recommend for 5.2.2.2 is Owner Assurance. This method may be described as:

- Owner Assertion -- The recipient receives assurance from the owner that the public key is valid.

This method requires that the recipient trust the owner. For instance, the owner may be in a contractual relationship with the recipient where the owner agrees only to provide valid public keys, and takes liability damages caused for invalid ones. This may well be a requirement in contracts with a certificate authority. Owner Assertion provides a very low level of assurance, as no explicit mathematical checking is performed, but it is better than no having assurance at all.

2. Integer factorization schemes

Techniques based on the integer factorization problem (e.g., based on the RSA algorithm) are appropriately awaiting the draft ANS X9.44 to become stable. In general, the model in the draft Recommendation can be adapted straightfowardly to support the RSA-based methods in ANS X9.44. The main design changes are likely to be the following:

* New key agreement class. The key agreement scheme in the draft ANS X9.44 is a variant of Shoup's RSA-KEM. It has one static key pair (the responder's) and no ephemeral key pairs. However, in place of the ephemeral public key, it has a ciphertext that is sent from the initiator to the responder. To match this scheme, I propose a new class, C(1,0), be added to the draft Recommendation. (None of the discrete logarithm schemes would fall into this class, since there is always at least one key pair per party, or at least two total.)

[On a related point: The term "key agreement" has a broader meaning in ANS X9.44 than in other places. In other places, such as the draft Recommendation, key agreement is usually defined as:

key agreement (1) = key establishment where both parties influence the keying material

In ANS X9.44, it is defined as

key agreement (2) = key establishment where neither party controls the keying material

The distinction is subtle, but essential to support the key agreement schemes in the draft ANS X9.44. "Influence" means that the keying material depends substantially on a value contributed by the party. "Control" means that the keying material is specified directly by the party. In key transport, the sender controls the keying material. In key agreement, neither party controls the keying material.

Although a party in key agreement can typically choose its input so that the resulting keying material has particular properties (e.g., last so many bits are 0), this is "influence", not "control". It is not possible for a party in key agreement to choose its input so that the resulting keying material matches a prespecified value.

In the schemes the the draft ANS X9.44, neither party controls the keying material, thus satisfying (2), but it is possible that only one party (the initiator) influences the keying material, thus not satisfying (1). However, this is still different than key transport, and I believe it is reasonable to keep the name key agreement.

Another possibility is to introduce a new name such as "key encapsulation", but it would not cover all the modes of the key agreement scheme in the draft ANS X9.44, some of which are "key agreement" in the usual sense. The concepts are close enough that a broadening of "key agreement", with an explanatory note, is probably easier than a new term.]

* Alternate key transport method. The title of 7.3 should be changed to "Alternate IFC-based key transport," and should specify the two choices for _wrapping_ keying material with a public key in the draft ANS X9.44: PKCS #1 v1.5 encryption and RSA-OAEP encryption. (The other key transport scheme in the draft ANS X9.44 is based directly on a key agreement scheme, following the model in 7.2. To support it, one only needs to change the title of 7.2 to "Key agreement-based key transport".)

* New primitives. Support for RSA-KEM, PKCS #1 v1.5, and RSA-OAEP should be added through appropriate new primitives.

* Alternate key derivation and MAC. If existing industry practice such as the TLS handshake is to be supported, the key derivation function / MAC from TLS should be added. This would only be for the purpose for compatibility with existing implementations, the same approach as in the draft ANS X9.44.

* Alternate "other input" for key derivation, and MAC data. The MAC data and the "other input" for key derivation should allow for different formats, at least those compatible with the TLS handshake as reflected in the draft ANS X9.44. (On a more general note, one may wish to look at whether any changes are necessary to align the draft Recommendation with the use of Diffie-Hellman in other standards including TLS and S/MIME.)

* Key confirmation without identity. The draft Recommendation currently allows key confirmation to be provided only by parties with a static key pair. To align with the draft ANS X9.44, both parties should be allowed to perform the key confirmation operations, regardless whether they have a static key pair. Although such confirmation cannot provide assurance of a party's identity if the party does not have a static public key, it can nevertheless provide assurance that (some) party possesses the established key, which may be helpful in defending against a chosen-ciphertext attack.

* Initiator authentication. If the initiator (or the sender, in a key transport scheme) does not have a static key pair, it should be possible for the party to provide assurance of its identity via appropriate use of a signature scheme. This is already specified in the draft ANS X9.44 for key agreement, although it is not clear how one would specify the service (nor whether it is necessary to do so) for key transport.

I plan to provide some text for consideration by NIST when incorporating the integer factorization schemes, reflecting the discussion above.

3. DHAES mode

Most of the schemes currently in the draft Recommendation are variants of Diffie-Hellman key agreement, where one party's public key is combined with the other party's private key for form a shared secret value, and a key is derived from the shared secret value. In the finite field case, this has a form such as

Z = DH (x, t)
KDF (Z, OtherInput)

where x is the responder's private key, t is the initiator's public key, Z is the shared secret value, and OtherInput is other information for key derivation.

For the security reduction to be "tight" when the responder's party's public key is static (e.g., in dhOneFlow), it appears to be important that the OtherInput include a representation of the initiator's public key. This concern, originating from Bellare and Rogaway's DHAES proposal http://grouper.ieee.org/groups/1363/P1363a/Encryption.html#paper3, has been discussed extensively in the development of IEEE P1363a, where it is called "DHAES mode". It is also reflected in the draft ISO/IEC 18033-2. I would recommend that similar attention be given in the draft Recommendation, and preferably that the OtherInput be required to include a representation of a first party's public key when the second party's public key is static, unless otherwise constrained by interoperability with existing implementations. (It is not fully clear to me whether and how this applies to other schemes beyond dhOneFlow and One-Pass Diffie-Hellman, which is where the issue has been raised in the past, but I can attempt to provide more detail if needed.)

Additional Technical Comments

p. 8, "MAC algorithm": In other usage, the MAC is considered as a single function rather than a family, and the key is an inputs to the function. I would consider the underlying hash function (if any) as the parameter rather than the key.

p. 9, "Statistically unique": This definition seems almost impossible to satisfy in practice, if it refers to the probability that an output is _ever_ repeated. If a random generator keeps track of all previous outputs, it can ensure that the probability of repeating any given n-bit output is less than or equal to 1/2^n (simply by not repeating at all). However, if it doesn't keep track, then over sufficient time, the probability that _any_ previous output is repeated must approach 1 (after a very long time). Perhaps what is meant is that the probability of the _next_ output matching any given previous output should be at most 1/2^n. "Equal to" would then correspond to the case that the generator doesn't keep track of previous outputs, and "less than" to the case that the generator does keep track (where the probability of a repeat is 0).

Still, unless the generator is truly random, the probability may not be strictly less than or equal to 1/2^n, there may be a small chance that the probability is higher (too small to detect with any known algorithm in a reasonable period of time). Thus, I'd prefer a definition more based on "indistinguishability": that for an adversary at the given security level, the generator cannot be distinguished from a truly random source.

p. 10, "Text1, Text2": The bit strings do not necessarily need to be sent, they could also be derived from the parties from other shared information (e.g., key usage).

p. 12, Sec. 4, para 1, line 3: "where both parties contribute": See discussion above re: the definition of key agreement in the draft ANS X9.44

p. 12, Sec. 4, para 3, line 2: The process for generating domain parameters would typically be used before, not during key establishment.

p. 12, Sec. 4, para 3, line 5: The process may be random, so that the same output will not be produced. In this case "equivalence" should mean that the distributions of outputs are indistinguishable (by an adversary at the given security level).

p. 13, Sec. 5.1.1.1: Please see RSA Security's comments on the Guideline document regarding key sizes.

p. 15, item 3 at top: Why would domain parameters become invalid? Is it that they are _believed_ to be valid at first, and later found not to be?

p. 15, para 2: Presumably this is related to the assumption that SHA-1 provides 160-bit security in some contexts, which is allowed within the Guideline document. If so, it should be stated explicitly that this assumption is being made. The text could even be re-worded to be more positive: the use of a hash function for assurance of domain parameter validity depends only on the one-way property of the hash function, so users should refer to that column of the table.

p. 16, Sec. 5.2.2.1, item 3: While the public key is typically generated from the private key in the FFC and ECC schemes, this is not the case in IFC schemes, and is not required for FFC/ECC. (In the FFC/ECC case, a party may develop the key pair by composing together previously computed values, such that both the public and the private keys are output.) I'd suggest a more general statement: "The owner generates the public/private key pair".

p. 17, Sec. 5.2.2.4, items 2 and 4: Where would a "trusted processor" fit into a typical architecture? Does this mean that there is a processor between the initiator and recipient? Or is the processor part of one of the parties' implementations, just not under control of that party?

p. 17, Sec. 5.2.2.5, para 1, line 6: Change "method 1", "method 2" to actual names of the methods, or say "Full Validation". It may be clearer to say "This method shall be used to obtain Recipient Full Validation and TTP Full Validation in 5.2.2.1, 5.2.2.2, and 5.2.2.4." Similar comment in 5.2.2.6, 5.2.2.7.

p. 20, Sec. 5.2.4.2, item 1: Although the recipient can enforce the requirement by not generating its ephemeral key pair until it receives the owner's static public key, is the intent also to require that the owner not generate ephemeral key pairs until after the static public key has been generated? This would be very complicated to enforce, since the owner could well generate all its ephemeral key pairs in advance. The requirement seems to be directed toward unknown key-share attacks on the MQV protocol, where the attacker replaces a party's ephemeral and static public keys with its own and to do this must see both public keys first before forwarding either of its own to the other party. However, the "time-ordering" countermeasure is only effective against attacks on the interactive MQV protocol, not store-and-forward. Also, the use of key confirmation and/or the inclusion of the parties' names in the input to the KDF can address the unknown-key-share attack. Are there other reasons for this timing requirement?

p. 21, item 2 at top: This would seem to discourage the use of precomputation, which can be helpful for implementation efficiency, though I don't disagree with the motivation.

p. 21, Sec. 5.3 and following: The specification of the key derivation function mixes together the algorithm and the inputs to the algorithm. These aspects should be kept separate if possible, so that implementations can distinguish between the specification of the KDF (the algorithm) and its usage (the inputs). It is easier to do this for the concatenation KDF (the preferred one) than for the ASN.1 KDF, since the counter is part of the inputs to the ASN.1 KDF and changes as the algorithm progresses. Also, there does not seem to be any high-level description of how key derivation fits into the key agreement process. I'd therefore suggest the following reorganization:

* Define only the algorithm in 5.3. This is the transformation from Z and OtherInput to DerivedKeyingMaterial. Ideally, only one algorithm should be given, supporting both the concatenation and ASN.1 KDFs. One way to do this is to avoid the counter entirely at this level, instead letting the application specify the counter. That is, the KDF would just be the hash function with a fixed-length output (or an output length bounded by the hash function length).

* Explain how the algorithm is used in Clause 6. This would include the general framework where a secret value Z is formed from the parties' keys and keying material is derived from Z (which is common to all the key agreement schemes), the requirements on OtherInput, and the text about zeroizing Z before outputting DerivedKeyingMaterial.

* The requirements on OtherInput should be broadened so that other formats can be supported. Currently, there are two choices:

OtherInput = U || V || SharedInfo

OtherInput = DER-Encoding (AlgorithmID || counter || PartyUInfo || PartyVInfo || SuppPrivInfo || SuppPubInfo)

To support the TLS handshake, the requirement should be relaxed so that other formats are allowed and so that identifiers for U and V do not need to be included (although I agree that they should be recommended in general). Indeed, the requirement needs to be relaxed in any case to support ephemeral-ephemeral DH, where the parties may be anonymous.

* So that can be just one KDF, I'd suggest that the first recommended format be changed to

OtherInput = counter || U || V || SharedInfo

and the KDF changed so that it no longer has a counter. In the schemes, instead of the statement "Compute KDF(Z, OtherInput)", one would have "Derive keying material from Z." This would result in a more flexible solution since the process for deriving keying material could involve multiple calls to the KDF, perhaps a different call for each key.

If the recommended key sizes and algorithms are followed, the output of the hash function will be _longer_ than the corresponding symmetric keys. So, it will rarely be the case that the KDF needs to be called more than once to generate a single symmetric key. In some applications such as TLS, the keying material is a long string that is later parsed into separate keys. But with just a minor impact on efficiency, one may generate the separate keys with separate calls to the KDF. This would make the generation more intentional and help to integrate the KDF better into the underlying key management --- each KDF output would directly be stored as a separate key, without further processing (e.g., parsing) by the application.

p. 22, 5.3.1, "Input", item 3: keydatalen _can_ (potentially) equal hashlen x (2^32 - 1), since j will be 2^32 - 1 in this case, same as with the current upper bound. Same comment in 5.3.2.

p. 22, Sec. 5.3.1, "Input", item 4: I'd expect hashlen to be a _parameter_ of the KDF, not an input, since it is related to the underlying hash function and not the key being derived. Likewise, the function H should be a parameter in 5.3. In other words, the KDF is parameterized by the choice of hash function, which shall be approved; hashlen denotes the length in bits of the hash function output. This is the same approach as in the draft ANS X9.44 for the function KDF2 (although there, the length is in octets, but that's a separate issue).

p. 23, para 2 at top: To provide abstraction, the KDF should output "invalid", not the scheme. (Same comment in 5.3.2.)

p. 24, Sec. 5.4, para 1, line 1: "one-way" --> "keyed one-way"; change "parameterized by a symmetric key" so that the symmetric key is an input rather than a paramter (the underlying hash function would be the parameter, as discussed above)

p. 25, Sec. 5.4.2, lines 3, 4: The inference that the MacKey and MacData values are the same is based (in general) on the assumption that the MacKey is secret. For HMAC, the inference would still hold if the MacKey were public but for other MACs (e.g., one-time unconditionally secure MACs based on

combinatorics), the MacKey has to be secret. (The MACKey is of course secret in this schemes, but it should be made clear that the inference depends on the secrecy.)

p. 26, Sec. 5.8 and following: The primitives should indicate their assumptions on validity (i.e., domain parameters assumed to be valid, public key assumed to be valid or partially valid as appropriate, private key assumed to be valid). Note that some of the "failure" conditions are superfluous if the assumptions on validity are met. For instance, FFC DH cannot produce Z = 1 if the keys are valid. ECC CDH, on the other hand, will produce P = O if the public key, which is assumed to be partially valid (i.e., on the curve), is entirely in the cofactor subgroup.

p. 51, Sec. 7.1, item 3: Is "manual" establishment necessary? Couldn't the key-wrapping key be established by other means, e.g., a previous key establishment scheme?

Also, more generally, should other symmetric methods be included, perhaps involving a Key Distribution Center? NIST has done considerable work on such methods in the past, and symmetric methods such as Kerberos are widely implemented.

p. 66, item 2 at top: Should EphemPubKey_V be included, if available from the key agreement scheme?

Editorial Points

p. 6, Sec. 1, para 1, line 9: "exponentially" --> "quadratically"?

p. 9, "Responder": The responder doesn't receive the keying material per se (both parties in key agreement derive it). I'd suggest "The second party in a key agreement transaction. Compare with initiator."

p. 10, "U" and "V": Suggest "ID_U" and "ID_V" for the bit-string identifiers, to avoid ambiguity with the parties themselves.

p. 10, "//s//": Italicize "s" after "bit string"

p. 11, "G": "distinguished point" --> "generator of subgroup order n"?

p. 12, Sec. 4, para 1, line 2: Add "(KT)" after "key transport"

p. 12, Sec. 5.1, para 1, line 4: Define "validity" before first use.

p. 12, Sec. 5.1, para 1, line 8: "multiple key establishment schemes" --> "multiple key pairs and with multiple key establishment schemes"

p. 13, para 1, line 1: "one scheme" --> "a given run of a scheme" (I consider the "scheme" to be the parameterized specification, an "instance" of a scheme to be the specification with given parameters, e.g., underlying hash functions, and a "run" to be a single execution of an instance of a scheme with particular inputs)

p. 13, Sec. 5.1.1.1, para 4, line 8: Define "transitive trust"

p. 13, Sec. 5.1.3, line 2: "destroy" --> "deactivated"?

p. 13, Sec. 5.2.2, lines 5, 6: "key agreement" -> "key establishment"

p. 16, Sec. 5.2.2.2, item 3: "receivesd" --> "receives"

p. 17, Sec. 5.2.2.5, para 1, line 1: Missing text at start of para

p. 17, Sec. 5.2.2.5, "Process", item 1: "Ensure" --> "Ensures" (compare 5.2.2.6)

p. 18, line 1: "Ensure" --> "Ensures"

p. 20, Sec. 5.2.4.2, item 6, line 6: "CA provides" --> "CA obtains"? (The distinction being that the CA has to first obtain the assurance itself before it provides the assurance through a certificate.) Similar comments elsewhere.

p. 21, item 3 at top, line 4: "the its" --> "its"

p. 22, "Process", item 4: "keylen" --> "keydatalen"; "keydatalen - ((hashlen) x (j-1))" --> "keydatalen mod hashlen" (for simplicity)

p. 22, para 2 ("Any scheme ..."): "for a bit string of length" --> "with keydatalen"

p. 22, Sec. 5.3.2, para 3: "maxhashlen" --> "maxhashinputlen" (and in other places, to distinguish from hashlen, which is an output length); shouldn't 5.3.1 have the same text?

p. 22, Note 1, line 3: "e an" --> "an"

p. 24, Sec. 5.4, para 1, line 4: italicize "MacTag"

p. 24, Sec. 5.4, para 2, line 3: extra "."

p. 25, Sec. 5.5, "Process", items 1, 2: introduce a new symbol for the converted x_P to avoid overloading

p. 27, Sec. 5.8.1.1, "Process", item 2: "1" should not be italicized

p. 28, item 2 at top: use special form of "O"

p. 28, Sec. 5.8.2.1: Add note that "second" keys may be ephemeral or static depending on MQV1 or MQV2 form, with pointer to 5.8.2.1.1 and 5.8.2.1.2

p. 29, Sec. 5.8.2.2, "Process", items 1, 2: use "S_A" instead of "implicitsig_A" for shorthand and consistency with FFC version? (in any case, italicize)

p. 34, Sec. 6.1.1, Figure 1: lines have different thicknesses; "compute a shared secret" --> "form a shared secret" (compare other figures)

p. 35, Table 7, "Computation" row": "Z_e" should be italicized

p. 37, Table 9, " Computation" row: Missing "," before "U's ephemeral public key"

p. 37, Table 9, "Derive Keying Material" row: "f" shouldn't be italicized

p. 39, Table 11, "Static Data" row: "Static Data" should be bold

p. 45, Sec. 6.2.1.5, para 2, line 4: "[SharedInfo]" is just for the concatenation KDF; different field names are used in the other KDF) (may affect text elsewhere)

p. 44, Table 15, "Input" row: "(p,q,g)" should be italicized

p. 45, Sec. 6.2.1.5, para 3, line 5: "responder" --> "responder with the corresponding public key" (here and possibly elsewhere) (the responder is not compromised if it uses a different key pair)

p. 47, Table 17, "Computation" row: "Z" should be italicized

p. 51, para 3, line 1: "to either party" --> "to either party or to both parties" (here and in other places where bilateral key confirmation is allowed)

p. 54, Sec. 8.1, para 1, line 4: "doe" --> "does"

p. 65, item 2 at top: "||" shouldn't be italicized

Date: Thu, 03 Apr 2003 16:03:09 -0500
From: John Kelsey <kelsey.j@ix.netcom.com>

Some Potential Flaws in the Proposed NIST-Approved KDF
John Kelsey
April 3, 2003

1 Summary

The basic KDF described in the draft of Special Publication 800-56 has some potential problems that may make it a bad choice for use as a general-purpose KDF, or that may at least require some changes.

The problems I have noticed include:

a. I think the Otherdata fields need to have some encoding rules specified, so that they are parseable. (It's possible that this requirement is documented somewhere else, and I just haven't seen it. In that case, disregard this point.) This would avoid both field-sliding attacks and length-extension attacks on the KDF, which appear to now be possible.

b. The length of the requested key should be included in the hash calculation. Because this isn't done in this KDF, it's possible for two sides to request the same key, with different lengths, leading to possible security problems.

c. The KDF is limited to hashlen bits of security, regardless of how many bits of random data appear in Z, and regardless of how many bits of derived key are requested. For example, using SHA1, a 256-bit AES key might be generated from a Diffie-Hellman key exchange blob of 15,360 bits. The result would be a key with only 160 bits of security, not 256 bits, because of this property of the KDF.

2 KDF Definition

The KDF takes the following inputs:

a. Z, which is assumed to be unknown to the attacker, but which may be reused for many different derived keys.

b. Otherdata, which is broken into several parts:

(i) U and V, identifiers for the entities deriving the key
(ii) keydatalen, the requested length of the key
(iii)hashlen, the length of the underlying hash function
(iv) SharedInfo, optional data shared by the parties

In order to generate a derived key, the KDF operates as follows:

j = ceil(keydatalen / hashlen)

for i = 1 to j:
    hash[i] = HASH(Z||i||U||V||SharedInfo)

The sequence of hash[0,1,2,...] values is returned as the derived key, with the last hash value truncated as needed.

3 Encoding Problems

Perhaps I've missed something in the document that explains why this isn't really a problem, but it looks to me like there's no specification of how the fields in Otherdata are encoded. They obviously need to be encoded in a way that's parseable, since otherwise, there could be different sets of values for those fields that ends up yielding the same key.

4 Output Length

The requested output length should be included in the hash calculations. When this isn't done, it's possible to get two different keys (perhaps for different algorithms) of different length, which share the first N bits of their value.

To see why this might be a problem, consider a request for a two-key 3DES key, and the same parameters used to request a single-DES key:

K1 = d4 1d 8c d9 8f 00 b2 04 e9 80 09 98 ec f8
K2 = d4 1d 8c d9 8f 00 b2

The 3DES key ends up being vulnerable to brute-force attack, by simply getting the other party with the shared key information to generate and use a single-DES key with the same information.

A properly-designed protocol ought to prevent this kind of attack in a number of ways (such as not enabling ciphers that can be broken with off-the-shelf technology). But a general-purpose KDF cannot depend on this.

This problem can be solved by including the requested key length in the hash.

5 Length Extension Attacks on SharedInfo

There is no reason to assume SharedInfo will not be chosen by an attacker for this KDF. But if SharedInfo isn't encoded properly, an attacker can do a length-extension attack on it.

That is, suppose I know a derived key of 480 bits from

(K1,K2,K3) = KDF(Z,U,V,hashlen,keylen,SharedInfo)

Then, using the length-extension properties of all widely-used hash functions (MD5, SHA1, SHA256/384/512, RIPEMD160), I can choose a new SharedInfo', with the property that if I know (K1,K2,K3), I can compute the new derived keys for that new SharedInfo. My new SharedInfo' just contains SharedInfo, plus the standard padding for the hash function being used, plus any other data I care to add.

The solution here is to encode the SharedInfo information in a way that prevents length-extension attacks. This can be handled along with encoding Otherdata intelligently.

6 Limits On Strength of Derived Key

Because of the way hash functions like SHA1 and SHA256 work, each different length of Z has a maximum amount of cryptographic strength its derived keys can have. This maximum amount bounces around in a rather unintuitive way, though it is never lower than 160 bits for SHA1.

Suppose Z is a multiple of 512 bits long. Then, a hash function like SHA1 will process the whole value of Z, and will get an intermediate chaining variable of 160 bits. That chaining variable is the only information about Z that makes it into the derivation of the key. Thus, even if Z is 2048 bits long, and we request 2048 bits of derived key material, the entire block of derived key material can be guessed with only 2^{160} work.

If Z is a different length, it is possible to get more security from the derived keys. Again assuming SHA1, the best possible level of security is achieved by having Z be 32 bits shorter than a multiple of 512 bits; in that case, the keys can get up to 572 bits of security.

Note how this works: The KDF processes the secret, Z, first. Suppose Z is 512 bits exactly. Then SHA1 processes Z, and computes a 160-bit chaining variable as a result. This same variable is computed for every one of the hash[i] values computed, since each processes Z first. So, an attacker who could guess that 160-bit chaining variable could predict every hash[i] value from the shared secret value Z, and thus would know every key that could be derived from Z with this KDF. This works because all other data besides Z is public; any attacker will likely know it all.

This problem is not specific to SHA1; it is an interaction of the way the KDF is specified and the way SHA1 hashes a whole string by hashing a 512-bit piece at a time with its compression function. All widely-used hash functions have this same structure.

This is surprisingly difficult to fix. If the counter were prepended to Z, then each derived key from Z could get the full entropy of Z, but given knowledge of one derived key from Z, an attacker could generally derive others with a 2^{160} guess (for SHA1).

I think a very reasonable solution for this problem is to simply put a note in the section about the KDF, explaining that it cannot reliably provide more than N bits of security, where N is the number of bits in the hash function used. Thus, the KDF with SHA1 is guaranteed to provide 160 bits of security, with SHA256, 256 bits, and with SHA512, 512 bits. Even with N=160, this is basically just an academic concern, since there probably aren't any real-world systems where a 160-bit keysearch is the easiest way to attack them.

I hope this is helpful,

--John Kelsey, kelsey.j@ix.netcom.com, April 3, 2003

--John Kelsey, kelsey.j@ix.netcom.com

From: "David A. Mcgrew" <mcgrew@cisco.com>
Date: Thu, 3 Apr 2003 13:46:48 -0800

Comments on the Key Establishment Schemes Document
David McGrew, Joseph Salowey, Paul Gleichauf, and Brian Weis
Cisco Systems, Inc.
April 3, 2003

The NIST Crypto Toolkit is a valuable program, and we are especially interested in its use in the Internet. We think that the scope and direction of the Key Management Project is correct. However, we are concerned that the proposed Key Schemes [1] deviate significantly from current Internet practice. Many existing cryptomodules used in Internet protocols would not be re-certifiable were this specification to be made normative. It would be a significant loss to the Internet community if TLS, IKE, Kerberos, SSH, and other IETF standards could not make use of the FIPS-140 program.

Moving forward, it is essential that we identify the differences between the NIST key management documents and the important IETF standards, identify the changes needed to both the key management documents and those standards, then work through those changes. We hope that NIST's crypto toolkit retains its inclusive position on key management until this work is completed, so that Internet users are not forced to choose between FIPS-140 conformance and IETF standards.

There are many important differences. Some of the differences represent technology opportunities for Internet protocols; for example, the use of elliptic curve cryptography. We support the adoption of these technologies and value NIST's encouragement of them. Other differences are more fundamental, such as the key derivation and key transport mechanisms. The key wrap mechanism [2] has no antecedent in Internet usage, and its intended use is unclear to us. That mechanism provides confidentiality and integrity protection, but appears to not provide anti-replay protection. It is redundant to use this mechanism in a key management protocol that already provides all three of those security services, yet seems insufficient to protect keys by itself since it lacks the third service. The public key mechanisms may be applicable in some cases, though this is not clear to us. The IETF culture, which expects free and easy access to specifications, has unfortunately inhibited the review and adoption of the ANSI standards referenced by the NIST documents within the IETF.

Some other differences are due to the fact that there are mechanisms in IETF protocols that are not addressed by the NIST key management documents, such as forward security, identity protection, and denial of service protection. These security services may not be as important as confidentiality. However, it would be regrettable if implementations of protocols that provided these services could not participate in the FIPS-140 program due to the incompatibilities between standards.

[1] NIST Special Publication 800-56: RECOMMENDATION ON KEY ESTABLISHMENT SCHEMES, Draft 2.0, January 2003. Online as keyschemes-Jan03.pdf.

[2] AES Key Wrap Specification, 16 November 2001. Online as key-wrap.pdf

Comments on NIST Special Publication 800-56 (Recommendation for Key Establishment Schemes, Draft 2.0)

Daniel Brown, Certicom Research

Main Comments

Firstly, I must congratulate and thank NIST for its excellent work on SP 800-56. The key establishment schemes NIST has decided upon so far provide a versatile palette of security and efficiency attributes, and yet also carefully delimit the variability, which will surely encourage interoperability.

I am hopeful that NIST approval of key establishment schemes will inspire wide-spread confidence and offer greater clarity in the choice and use of public-key based key management, which up to now has been somewhat of an *ad hoc* endeavour.

Further Comments

To assist in NIST in its efforts to finalize SP 800-56 in a timely fashion, I offer some further, fairly insignificant comments.

The categorization of the following comments into general, technical, editorial and typographical is intended to ease their processing.

General Comments

Ordered roughly according to importance.

1. This Special Publication does not indicate that it will become a FIPS document. Because major crypto decisions of NIST were embodied in FIPS document, the appearance that this document is not destined to be a FIPS might cause confusion at first.
2. The schemes in this recommendation will presumably become mandatory for key establishment, regardless of the document ends up as a FIPS or an SP.
3. At present it is not possible to validate or vendor self-affirm the algorithms in this draft recommendation, and furthermore the draft does not indicate if the vendor self-affirmation will be available as interim measure until the validation systems are available or if the validation systems will be available by the time the recommendation is finalized.
4. It is pleasing to note that the key agreement schemes can be transformed into key transport schemes, which is consistent with the principles adopted S/MIME's CMS in RFC 2630 and RFC 3278.

Technical Comments

Ordered roughly in order or application to the draft recommendation.

1. Page 7, "Identifier". Can this be the bit string of length zero?
2. Page 13, 5.1.1.2. Is the cofactor "h" a mandatory component of EC domain parameters? (Compare to X9.62 and X9.63 where it is optional.)
3. Section 5.1.1.2, page 14. What's minimum size for "q" or for "n"? (Presumably, it's 2^{159}+1 more.) I'm guessing that SP 800-56 defers to SP 800-57 on this matter, and this is why SP 800-56 does not use a bold "shall".
4. Section 5.1.1.2, page 14. "May" elliptic curve domain parameters other than those in FIPS 186-3 be used? Presumably yes, or you would have set the cofactor maximum to be 4.
5. Page 31, Table 5. Why not include a C(2, 1) scheme? The responder would have a static and ephemeral key. The initiator would have just an ephemeral. The initiator is anonymous, as in C(1, 1). The responder has forward-secrecy upon compromise of its static key, as in C(2, 2).
6. Page 33, last paragraph before Section 6.1. Having fewer passes is even more important in systems that have high latency and limited interactivity, that is, if each pass incurs a delay because of a network queue or hopping process, then minimizing passes desirable.

Editorial Comments

Ordered roughly according to application to the draft recommendation.

1. Page 6. "… grow exponentially". The number of share symmetric keys is proportional roughly to the square of the number of entities, so "quadratically" would be more accurate than "exponentially". But since "quadratically" has not fallen into the common vernacular (as "exponentially" has, with serious degradation of meaning), and this introductory paragraph is intended for a fairly diverse audience, a more sensible substitute should be chosen, such as "rapidly", "exorbitantly", "excessively", "explosively", "dramatically". Although some readers might have been accustomed to the abuse of the term "exponentially" (which is mathematically meaningless in the short term), we should not pander to these readers in this document because we risk undermining our credibility with technical readers who are likely to have heard of the n^2 phenomenon and who would be bewildered by the additional exponential cost. More directly, are you saying that it is cryptographically infeasible today for 80 entities to distribute symmetric keys without using public key cryptography?
2. The current title of ANSI X9.44 is *Key Establishment Using Integer Factorization Cryptography*, not *Key Agreement and Key Transport using Factoring Based Cryptography*,
3. Page 8 & 9: "Receiver" and "Recipient" and "Responder". ANSI X9.63 uses "initiator" & "responder" in both key transport and key agreement. SMIME uses "receiver" and "recipient" as follows: "receiver" refers to the entity that received a cryptographic communication and "recipient" refers to the entity for which a cryptographic communication indicates it is intended. As a non-cryptographic analogy if Alice receives a snail mail addressed to Bob, she is the receiver and he

is the recipient. The correspondents on the opposite end are, "sender" and "originator", respectively. This distinction is useful both in discussing attacks where the receiver and recipient are different or the sender and originator are different and in discussing the actions of the sender and the receiver. For example, the sender operates on the recipient's public key (not knowing for certain who will receive the actual crypto data) and the receiver operates on the originator's public key (not knowing for certain who actually sent the crypto data).

4. Page 9. "Statistically unique". This definition is quantitative, so presumably it has a mathematically rigorous intention. To make sense, the probability is presumably taken over consecutive generation of two n-bit quantities, although the wording does not make this entirely clear. (To take the extreme, 2^n+1 generations of n-bit quantities would contain a "repeat" with probability one.) Under this interpretation I'm not sure how useful the definition is. Alternating back and forth between two values would set the probability of two consecutive generations repeating to zero, but I doubt this attribute contributes much to security. Another troubling aspect of this definition is that if want to generate uniformly at random (n-bit) quantities k satisfying $0<k<m<2^n$ then ideally the probability of two consecutive values repeating is 1/m which is larger than $1/2^n$. A simple search of the document for "Statistically" shows that "Statistically unique" is only applied to values that are generated by an "Approved" random number generator. We can assume that such a random number generator will provide the necessary security. Therefore, it is not necessary to further qualify the Approved random number generator as "statistically unique", although I suppose that it does not harm. (Or perhaps, I'd stand corrected, if some not "statistically unique" random number generators are Approved, then one might indeed to qualify in this standard that the Approved RNGs that are "statistically unique" should be used.)
5. Page 13, 3rd paragraph from bottom. After "provide more security assurance", insert "than shorter FFC keys" because it could be mistaken as comparison to other kinds of keys. Add similar insertion for ECC keys on Page 14.
6. Page 13, next paragraph. Change "an exact length for a specific" to "a single length for each specific".
7. Page 14, 1st paragraph, "where p is an odd prime" and "where m is an odd prime". The word "odd" is hardly helping much because it only excludes one prime, 2. Presumably, the minimum size for "q" should ensure that neither p nor m is 2 (or any prime that is too small). This begs the question, what is the minimum size for "q"?
8. Page 14, 2nd paragraph. Change "specifies recommended elliptic curves" to "specifies recommended elliptic curve domain parameters".
9. Page 14, Section 5.1.1.2, 2nd sentence. Change "the cofactor is" to "cofactor multiplication is".
10. Pages 14 & 15, steps 2 a & b. What is "keysize check"? Is this checking the bit lengths meet the requirements stated in 5.1.1.1 and 5.1.1.2?
11. Page 18, Step 4, Comment. Remove "is in the correct range" which does not make sense.
12. Page 20, Steps 2 & 3. Isn't Step #3 assured by Step #2.
13. Page 22, Output. Insert "length" between "of" and "keydatelen".

14. Page 23, Step 3.1.1. Presumably such object identifiers are available somewhere?
15. Page 25, Section 5.5. "P" is referred to as a point, and then its "validity" is discussed, although technically validity is applied to public keys. Why not call P a public key?
16. Page 25, Process, Step 1. It might help to use a separate notation for x_P and the integer it is converted into.
17. Page 26, Section 5.8, 2nd line. Change "as such" to "so as".
18. Page 41, last sentence before 6.2. An explanation why key confirmation cannot be added to C(2, 0) schemes would be helpful.
19. Page 48, 2nd last paragraph. Change "identifier" to "identity".
20. Page 54, 1st paragraph. Shouldn't "q" be change to "p" in both places?

Typographical Comments

Ordered roughly according to application to the draft recommendation.

1. Table of Contents. (Page 3). The five-deep section #'s are indented more to the left than the four-deep section #'s and 5.8.2.2.2 uses an inconsistent font.
2. Table of Contents. (Pages 3-5). In C(#,#,XYZ ABC), spaces should follow commas.
3. Page 8, "MAC". Delete "Defines", because this defines an object as an action.
4. Page 10, "MQV". When I print or view the PDF the word "agreement" in this definition has the "n" and "t" too close together. Similar problems occur throughout the document, some letters are too close and some are too far. This problem may depend on the machine viewing the PDF file, perhaps it has to do with the fonts used. I'll provide just a few more of these typos, with a more extensive list available upon request.
5. Page 10, "MQV". I'd think MQV should be in roman font, not italics.
6. Page 10, //s// and X||Y. The slash risks looking like an italics vertical line, and vice versa.
7. Page 10, "mod p". Change "on" to "of".
8. Page 10, "rU, rV". Put second "r" in italics.
9. Page 12, first sentence. (?) Change i.e. to "that is", add "(KT)" after key transport.
10. Page 12, Clause 5, last sentence, "calculations", has "i" and "o" too close.
11. Page 13, Table 1. Bottom border line of table missing.
12. Page 13, 2nd last paragraph. Change "[3] will also" to "FIPS 186-3 [3] will also". Change "trust, or" to "trust or" and change "trust in a" to "trust, in a". Same thing on page 14.
13. Page 15, footnote 1. Change "FIPS 186-2" to "FIPS 186-3".
14. Page 17, Section 5.2.2.5. Change " candidate FFC public key" to "A candidate FFC public key".
15. Page 18, Step 3. Put "a" in second equation into italics.
16. Page 19, Step 3. Put each "a" in two equations into italics.
17. Page 20, Step 1, Note. "is" has "i" and "s" too close.
18. Page 22, Process, Step 4. Change "keylen/hashlen" to "keydatalen/hashlen".

19. Page 22, Note 1. Remove "e" after "e.g.".
20. Page 23, Step 3.1.2. The initial value should probably put the last two digits "16" in subscript. (Otherwise the counter appears to start from the integer 278.)
21. Page 28, 1st line. Font for O not the same as used in other places.
22. Page 28, Process, Step 1. Change "||q||" to "//q//". Change "| … |" to ceiling symbols.
23. Page 30, Section 5.8.2.2.2, Heading. Change "MQV MQV" to "MQV". Sans serif font is not consistent with Serif font of Headings 5.8.2.1.1, 5.8.2.1.2, and 5.8.2.2.1.
24. Page 35, Table 7, Computation. Adjust fonts in "Z_e || Z_s" to 'Z_e || Z_s".
25. Page 40, Section 6.1.2.3. 1st line: "schemes" has "h" and "e" too close and next paragraph "isolated" has "a" and "t" too far.
26. Page 41, Figure 3, last line (both Steps #2). Bottom of characters is clipped.
27. Page 44, Table 15, Input. Put "(p,q,g)" in italics.
28. Page 52, last paragraph before 7.3. Number steps should be capitalized, e.g. "Step 2".
29. Page 54, 1st line. "F" should be in italics.
30. Page 54, Section 8.1, 4th line. Change "doe" to "does".
31. Page 65, Step 2, Change "02 // P // R || …" to "02 || P || R || …".

From: "Walker, Jesse" <jesse.walker@intel.com>
To: GuidelineComments@nist.gov
Date: Thu, 3 Apr 2003 08:52:47 -0800

This is already a nice document, so my comments are limited.

The documents contains numerous references to SP 800-56. Clause 5.3 of SP 800-56 defines key derivation functions. The key derivation functions defined in Clause 5.3 of SP 800-56 are fine, but the list of approved key derivation functions seems deficient. Here are two suggested additions for you to entertain:

a. In order to be compatible with commercial standards such as IPsec, it would be useful if an HMAC-SHA based key derivation function were also approved. This would be similar to the "3.5.1 concatenation key derivation function (Default)":

1. initiate a 32-bit big-Endian counter as 0x00000001
2. j = ceiling(keydatalen/hashlen)
3. for i = 1 to j by 1, do the following:
    3.1. compute Hash[i] = HMAC-H(Z, counter || U || V || [SharedInfo])
    3.2. increment counter
4. let Hhash = Hash[j] if (keydatalen/hashlen) is an integer, otherwise let
   Hhash = (keydatalen - (hashlen * (j-1)) leftmost bits of Hash[j]
5. set DerivedKeyMaterial = Hash[1] || Hash[2] || ... || Hash[j-1] || Hhash

where Z, U, V, and SharedInfo are as in the original. The allowed values for H would be one of the allowed values from Cluase 5.6 of SP 800-56.

b. To facilitate implementations in some constrained environments, it would be useful if a key derivation function based on AES-CBC-MAC were defined as well. The suggested key derivation function would be similar:

1. initiate a 32-bit big-Endian counter as 0x00000001
2. j = ceiling(keydatalen/hashlen)
3. for i = 1 to j by 1, do the following:
    3.1. compute Hash[i] = AES-CBC-MAC(Z, counter || U || V || [SharedInfo])
    3.2. increment counter
4. let Hhash = Hash[j] if (keydatalen/hashlen) is an integer, otherwise let
   Hhash = (keydatalen - (hashlen * (j-1)) leftmost bits of Hash[j]
5. set DerivedKeyMaterial = Hash[1] || Hash[2] || ... || Hash[j-1] || Hhash

where again Z, U, V, and SharedInfo are as in the original, except Z must be constrained to 128, 196, or 256 bits in length.

Jesse Walker
Intel Corporation

JF3-206
2111 N.E. 25th Avenue
Hillsboro, OR 97124
(502) 712-1849
jesse.walker@intel.com