Comments Received on Draft Special Publication 800-56 Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography

Summer, 2005

From: Daniel Brown <DBrown@certicom.com>
Date: Thu, 14 Jul 2005 17:04:20 -0400

Some early comments on private key generation. (More comments may follow.)

Section 5.6.1.2 and Table 2 are essentially contradictory: the bit length of a *random* integer d in [1,n-1] is generally not as given in Table 2. It is therefore impossible to comply with 5.6.1.2. The bit lengths in Table 2 generally produce d that are much smaller than n (and potentially even produce d>n). Use of d with of bit length shorter than n, as specified in Table 2, must be discouraged because of attacks like Bleichenbacher's or Leadbitter-Smart on MQV with partially known nonces. (One interpretation of 5.6.1.2 is that you purposefully intend to choose a biased d in the interval in [1,n-1], which would quite surprise me given the past problems with DSA.)

Suggestion: remove the requirements on the bit length of the private key, at least for the ECC case. Your other requirements, such as d being unpredictable, in the range [1,n-1] and the output of an Approved RBG should suffice for security.

Strictly speaking, 800-56 has not defined bit length of an integer, (I presume an ECC private key d is an integer from page 20). (Bit length is only defined for a bit string, and no fixed representation of an integer as a bit string has been defined.) So, not is only is 5.6.1.2 contradictory in spirt, it is not fully defined. It's actually ambiguous too: for example, what is the bit length of 5? Is it 3, as in "101"? Or do you imagine representing it as a bit string of length 160 in the form "0000....00101" in the FA and EA settings? I guess the intention was the latter (for if it was the former, then d would generally be much larger than 1 in addition to being smaller than n.) This problem with 5.6.1.2 and Table 2 will be resolved by the suggestion above.

Dan Brown
(905) 501-3857
http://www.certicom.com

Date: Mon, 18 Jul 2005 18:00:34 -0400
From: Stephen Kent <kent@bbn.com>

Congratulations on getting this eagerly awaited document out!

I was skimming the document and noticed a typo:

- the text uses "Certificate Authority" instead of "Certification Authority" in three places, but uses the latter term in three other spots :-)

Steve

From: tfletcher@ca.safenet-inc.com
Date: Thu, 21 Jul 2005 13:27:52 -0400

I have done a fairly quick initial review of the latest draft of SP 800-56 and I have a few comments. In general, I appreciate the efforts of you and your colleagues to pull this document together. It is badly needed.

My comments follow:

Comment #1

References:

Citing SP 800-57 (Draft) in Sections 2, 5.5.1.1, 5.5.1.2, 5.6.4.1, 7, 9

Comment:

I am concerned that making reference to the draft SP 800-57, especially in cases where the reader is directed to SP 800-57 for guidance, might delay final acceptance and approval of SP 800-56. From a vendor's perspective, I perceive SP 800-56 as the more important document, since it specifies cryptographic algorithms and mechanisms that can be implemented and tested. SP 800-56 appears to be further advanced than SP 800-57 and it would be unfortunate to have its approval held up pending finalization and approval of SP 800-56.

Comment #2

References:

Citing AES Key Wrap in sections 7, Appendix A

Comment:

I would like clarification on the status of the AES Key Wrap Specification. FIPS 140-2, Annex D lists AES Key Wrap as being a Draft and CMVP does not have a validation test suite for it. Has the AES Key Wrap Specification been fully approved? If so, is there any indication when the CMVP documentation and test suites will be updated accordingly?

Comment #3

References:

Section 5.8.1 and Appendix B

Comment:

I am not sure I agree with the idea that the counter values are the most variable. From one use of the KDF to another, the shared secret is actually more variable (and unpredictable) than the counter value. If the motivation is to ensure that the maximum number of bits of the input are involved in the mixing done by the hash function, then I believe that Z should stay as providing the initial bits in the input.

Comment #4

References:

Appendix B.5

Comment:

I am not clear why the Warning is given in Appendix B.5 regarding the use of the TLS PRF. Is it because of the use of MD 5? Is it because of the overall structure of it - splitting the pre-master secret in half computing two separate KDF results and XORing them? Or, is it because of the internal structure of the PRF itself - i.e., HMAC (SeedKey, indexi || FixedString), where indexi = HMAC (SeedKey, indexi-1)?

Related to this, why isn't it simply the KDF by itself, rather than the whole TLS scheme, that is referenced in this document? The use of the KDF in TLS could then be described as a particular case. I would prefer to see this approach taken since the KDF is generalizable to other applications and to use other digest functions, such as SHA-256. Has any consideration been given to this?

Comment #5

References:

Sec 4.1 para 4

Comment:

The reference paragraph starts, "This Recommendation assumes that there is a trustworthy binding of each entity's identity to the entity's static public key. The binding of an identifier to a static public key ...". The previous paragraph made a point of distinguishing between identity and identifier - which term is intended in the above quoted text?

Once again thanks for your good work in making this happen.

Best regards,

Terry Fletcher
Senior Security Architect
SafeNet Canada, Inc.
One Chrysalis Way
Ottawa, Ontario
Canada
K2G 6P9
(Phone) +1 613 723-5077
(Fax) +1 613 723-5078

Entrust Comments on NIST SP 800-56 Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (July 2005 Version)

21 July 2005

This document is a welcome addition to the NIST cryptographic toolkit. We support NIST's provision of guidance on the implementation of key establishment schemes. Entrust's particular comments on the document are as follows:

1. Throughout this document, references are made to the forthcoming FIPS 186-3. Since the FIPS 186-3 draft document is not yet available, it is difficult to thoroughly review this draft of SP 800-56. We may have additional comments because of this once FIPS 186-3 is available. We note with curiosity that according to column FC in Table 1, it appears that DSA in FIPS 186-3 will need to be able to generate a *p* of 2048 bits and a *q* of 256 bits even though these lengths are not assessed at the same security level.

2. General: There is a general lack of rationale for many of the choices made in this document. Inclusion of a short rationale or discussion explaining the choices made would help the reader understand and appreciate the choices made when other apparently valid options exist.

3. Section 5.2.1: In the description of the *MacTag* computation it should be pointed out that if the *MacLen* or *MacKey* length is incompatible with the MAC algorithm chosen, then an error should be returned.

4. Section 5.5.1.1: Background: (A) In the April 2005 draft of SP 800-57 Part 1, in Table 2 page 62 FFC primes are allowed up to 15360 bits in order to match the security level of 256-bit AES. This table is also being used by ANSI X9. NIST, Entrust and others participate in ANSI X9 and by consensus have agreed publicly with this table and the security analysis behind it. (B) Entrust comments for the previous draft of SP 800-56 requested consideration that 4096-bit and 6144-bit DSA-DH/RSA keys be added to the existing set of 5 key sizes of 1024, 2048, 3072, 8192 and 15360 bit keys.

In the July 2005 draft of SP 800-56, in Table 1 page 30, the FFC prime *p* is limited to 2048 bits. Furthermore, the column FC uses values that are consistent with having 128 bits of security, except for the prime *p* which has 112 bits of security, according to Table 2 of SP 800-57. When using this table, no FFC method could be used to protect data beyond 2030, this means that FFC methods appear to be being phased out.

No rationale has been provided for this apparent decision. It is true that ECC methods are expected to have better performance than FFC methods as key sizes increase, but there may also be applications where the performance impact of

using larger FFC primes is acceptable. This should be left as an engineering decision and thus Entrust has concerns with limiting key establishment primes to 2048 bits.

5. Section 5.5.2 (and throughout the document): The last sentence of this section contains a requirement that the party receiving assurance of domain parameter validity must "know which method(s) of assurance were used". It is unclear what is meant by this requirement. In particular, the use of the word "know" may cause confusion. Is this meant to imply that the user of an application that performs a key establishment operation must be informed of the method of assurance? Or, does the application have to take different action depending upon the method of assurance? Is it permissible for the method of assurance to be hardcoded and thus explicit knowledge is not required? Due to its ambiguity, this appears to be a difficult requirement to enforce. Similar requirements appear throughout the document and this comment applies to each occurrence.

Thus, we recommend replacing the last sentence of Section 5.5.2 with "The application performing the key establishment on behalf of the party **must** decide whether or not to allow key establishment based upon the method(s) of assurance that was used. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself."

6. Section 5.6.3.2.1: In order to obtain assurance of possession of a private key, it is not enough to simply perform key confirmation with the claimed owner serving as the key confirmation provider, as is described in this section. It is also necessary to ensure that the *IDP* field in the key confirmation calculation (the identifier of the provider) matches the identity that has been bound to the public key. In Section 4.1 it was assumed that each public key was reliably bound to an identity, thus we know that such an identity exists.

7. Section 5.6.3.2.2: As with the above comment, it should be mentioned with respect to Explicit Key Confirmation with a Trusted Party that the *IDP* field in the key confirmation calculation must match the identity bound to the public key. In this situation though, the binding will most likely occur through some out-of-band method (e.g., telephone verification of public key hash with owner).

8. The Explicit Key Confirmation with a Trusted Party option requires that key confirmation be performed as described in Section 8 of the SP. However, most Certification Authorities do not implement this particular flavour of key confirmation in order to provide assurance of possession. For example, the PKIX CMP (RFC 2510) describes equally valid methods of providing assurance of possession. Thus, Certification Authorities that follow common industry standard protocols for providing Proof-of-Possession will not be able to provide this service to their relying parties that conform to this SP. It would therefore seem reasonable that other methods of providing assurance of possession, for example

as specified in RFC 2510, be allowed here. Specifically, the methods of Section 3.2.8 of RFC 2510 should be allowed. Otherwise, Certification Authorities will not be able to incorporate the method described in this SP, since it is inconsistent with the certificate management protocols in use, and thus will not be able to provide assurance of possession conformant to this SP.

9. Section 5.7.2.1.2: In the last line of the second paragraph, "(*yA*)" should be "(*yB*)".

10. Section 5.8.2: The ASN.1 schema for OtherInfo should either be provided here or pointed to. Otherwise, the wording may be interpreted as any ASN.1 schema that includes these values is sufficient.

11. Section 6.3.1: It would seem reasonable in Action 1 to allow the nonce to be agreed upon in some other way instead of requiring that it be sent from *U* to *V*. For example, it could be a timestamp or counter value.

12. Section 7: In the first paragraph after the numbered list, it should be mentioned that C(0,2) schemes are also such that the receiver does not contribute an ephemeral key pair to the calculation of the shared secret.

13. Section 8: No rationale has been given for the requirement that key confirmation conformant with this standard can only be provided with respect to static public keys. It would appear that ephemeral public keys that have been bound to an identity could also be subject to key confirmation.

14. Section 8.2: In Step 4, the nonce (*NonceP*) should be sent as well, if necessary.

15. Section 8.4.4: It should be allowed that *NonceV* need not be sent from *U* to *V* if it has already been agreed upon. For example, it could be known to be a timestamp or counter. A similar comment applies to Section 8.4.6.

16. Section 8.4.5: In this section *EphemDataV* is defined to be NULL. However, it should be allowed to be a nonce, in order for *V* to have assurance that the *MacTag* cannot be used again. A similar comment applies to Section 8.4.7 and 8.4.9.

17. Section 8.4.8: The *NonceU* is not mentioned is the first paragraph of this section (as was done in other sections) and not included in Figure 16.

18. Section 8.4.10: It should be mentioned in the first paragraph that *U* shall contribute a nonce (*NonceU*), as was done for *V*.

19. Appendix A: It should be noted in point 1 that "ANSI X9.42 defines *MacData* **for validation testing** as "ANSI X9.42 Testing Message".

In point 10 it should also be noted that the counter appears before the shared secret in this SP whereas it appears after the shared secret in the ANSI standards.

20. Section B.4: There does not appear to be a generic KDF described in Section 5.8.

21. Section C.1: In Step 2, *Oi* should be replaced by *Oi.*

Typographical Error
1. Section 5.5 First sentence has an extra comma after “is”.

Date: Wed, 27 Jul 2005 18:05:17 +0200 (CEST)
From: Serge Vaudenay <serge.vaudenay@epfl.ch>

I would like to raise the point of seeds in domain parameters. Actually, most of existing signature schemes (DSA, ECDSA) provide an insufficient assurance of random selection when the seed is used.

Typically, the presence of the seed should assure that domain parameters were randomly selected. As I have shown in the following references, this assurance is not appropriate. A malicious user could indeed first select malicious domain parameters then pick a seed for which the validation scheme is correct, or even modify parts of the domain parameters which are not generated from the seed.

The Security of DSA and ECDSA
Advances in Cryptology PKC'03, Miam, Florida, USA, Lecture Notes in Computer Science No. 2567 et pp. 309-323 Springer-Verlag, 2003.
http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Vau03a

Digital Signature Schemes with Domain Parameters
The 9th Australasian Conference on Information Security and Privacy - ACISP, Sydney, Australia, July 13-15, 2004, LNCS 3108, pp.188-199
http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Vau04b

I hope this comment will help.

Best regards

Serge Vaudenay

Professor Serge Vaudenay Add: EPFL / IC / LASEC
INF 241 Station 14
Swiss Federal Institute of Technology (EPFL) CH-1015 Lausanne
School of Computer and Communication Sciences Tel: +41-21-693-7696/7603
Security and Cryptography Laboratory Fax: +41-21-693-6870

From: "Lev Drannikov" <ldrannikov@hotmail.com>
Date: Fri, 05 Aug 2005 12:33:14 +0000

I am surprised how many references onto your recent publication are in the Internet. It means the subject is very important and actual. It's time for this. Thank you.

Here, I want to say a couple of preliminary remarks to improve the draft.

1: The document is large enough, so a table of contents (bookmarks in pdf) is necessary to make navigation and references more convenient.

2: There exist a lot of other algorithmically insoluble problems (in addition to the discrete logarithm one) to be used in cryptography. For example, if we have a number of equations less than a number of unknown entities, it's impossible to determine all of them at the same time and context.

3: There exist other finite fields, not mentioned in your proposal. For example, I would be glad to discuss my achievements in this area (See attachment please). They were partly revealed to Communication Security Establishment, Department of National Defense in Ottawa, Canada, in 1999 - 2000. (A main part of a that time report was given them for comments.)

Thank you again for your great work and scrutiny.

Best regards,
Lev Drannikov,
Canadian Data Security,
Co-owner, Inventor.

Comments on Draft Special Publication 800-56,
Recommendation for Pair-Wise Key Establishment
Schemes Using Discrete Logarithm Cryptography

Hugo Krawczyk_

August 10, 2005

Abstract

This note contains a response to the request for comments concerning NIST's document in the title. We believe that the standards specified in this document need to be thoroughly revised to reflect the current knowledge and understanding of key establishment protocols. Problems with the current proposal include the insecurity of some of the specified mechanisms, the lack of rationale (formal or informal) for many of the design choices, the use of unnecessary or inappropriate techniques, and the unnecessary complexity of some of the specification. Fortunately, using existing and well-analyzed mechanisms all of the above problems can be resolved. We urge NIST (and the industry at large) to adopt the more secure, more systematic and more efficient alternatives pointed out here.

1 Introduction

The document "Draft Special Publication 800-56, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography" [1] (referred to here as 800-56) describes a number of mechanisms for cryptographic key establishment (KE), and related methods, based on discrete logarithm cryptography. The covered key establishment schemes are variants of two main protocols: MQV [14, 11] and the so called "unified model" (UM) protocol. Both belong to the family of implicitly authenticated Diffie-Hellman protocols [13] and are presented in several flavors (or modes) in 800-56. Specifically, each protocol has a 1-message, 2-message and 3-message variant, each with a different specification depending on whether the underlying group is an elliptic curve (ECC) or a finite field (FFC) subgroup of prime order. Additional mechanisms specified in 800-56 include key confirmation, key distribution functions (KDF), public key validation, and proof of possession (of a private key).

Unfortunately, some central mechanisms defined in 800-56 suffer from security weaknesses of different degrees, including the well-known insecurity of the "unified model" (UM) protocol (shown in [3] to be open to interleaving and known-key attacks) and the recently reported weaknesses of MQV in [10]. The 800-56 document would seem to predate much of the research work from the last decade that has resulted in significant advances in our understanding of KE protocols, their design and analysis. The main driving force behind this spec seems to be compliance (or "backward compatibility") with the very similar ANS X9 standards (that seems a plausible explanation to the fact that 800-56 standardizes protocols that the editors themselves showed to be insecure – as in the case of UM).

Whatever the reasons for the current specification, we believe that 800-56 must be thoroughly revised to reflect the current knowledge and understanding of key agreement protocols. Not only will this result in essentially better security, founded on modern

formal models, but will also provide an opportunity to simplify and streamline much of the specification, in particular by dispensing of unnecessary safety margins and by supplying full rationale to each design component.

If existing mechanisms cannot be abandoned immediately, then at least they should be phased out as soon as possible (similarly to the current plans for SHA-1, but with the advantage of a well-founded theory to back the choice of KE schemes). Hopefully, a revision of 800-56 will result in a better and more secure practice of cryptography not only by US agencies but by the industry in general.

2 Key Agreement Protocols

We start by considering the core mechanisms specified in 800-56, namely, the key agreement protocols. There are two basic schemes: unified model (UM) and MQV. Each is presented in its core form as a 2-message implicitly (mutually) authenticated DH protocol and also with two variants: a 3-message protocol that includes explicit key confirmation (KC) and as a one-message protocol for store-and-forward scenarios. Furthermore, each of these variants have two specifications depending on whether the protocol uses FFC or ECC groups. One visible drawback of the specification is that the protocols in the two algebraic settings have significant differences beyond the normal adjustments needed to represent the operations in the two settings (such is the case, for example, in the steps for computing the shared keys or the way public keys are validated). This has the effect of making the specification, implementation and analysis of the protocols more cumbersome without any substantial gain (and the document offers no clue as for the rationale behind these differences). As we show here, more secure and more uniform protocols can be defined with less complexity. To add to the confusion, the names of the ECC and FFC variants of the same basic protocol differ significantly (for example, one protocol is called "full unified model" in the ECC setting and "dhHybrid1" in the FFC setting). We take the liberty of referring to the protocols in a more unified way in this note.

Notation and conventions. We use exponential notation and terminology (as in the FFC case) but the same comments apply (unless otherwise stated) to ECC additive groups. We denote the generator of the subgroup over which the protocol is defined by g and its (prime) order by q (this is consistent with the FFC notation in 800-56 but not with the ECC notation where the order of the subgroup is denoted n – in the ECC case 800-56 reuses q to denote the order of the underlying ECC field). We refer to elements x 2 Zq as DH exponents and to group elements of the form gx as DH values. These values may be ephemeral if used for a single key exchange, or static if used as static keys. We use the following conventions that differ from the notation in 800-56: we use x, y to denote ephemeral exponents used by the parties and a, b to denote their static private keys.

For concreteness we provide most comments for the core 2-message variant of the protocols but most also apply to the other variants. In all these cases the two messages exchanged consist of ephemeral Diffie-Hellman values with the possible simultaneous or prior exchange of long term public keys and corresponding certificates. All these protocols determine a shared secret which is a key from which further keying material is

derived using a defined KDF (key derivation function). The schemes differ mainly in the way the shared secret is determined but also in some other aspects (such as the form of public key validation).

2.1 Unified Model

The UM (unified model) protocol (called dhHybrid in the FFC context) computes its shared secret as the concatenation of two values: Zs and Ze. The first is obtained as the DH computation applied to the pair of static public keys (i.e. Zs = g^{ab}) and the second as the DH computation applied to the pair of ephemeral DH values (i.e. Ze = g^{xy}). This protocol should be considered as plain insecure by any modern measure of key agreement security. Indeed, as shown in [3] the protocol is open to trivial interleaving and known-key attacks. We can think of no reason to standardize on a broken protocol (especially that these security shortcomings of UM are well known and easily avoided).

The protocol specification includes additional elements such as the requirements for proof of possession of private keys, validation of public keys, and, in the ECC case, the use of co-factor in the key computation; however none of these help solving the inherent weakness of the protocol. Such a weakness comes from the fact that the 2-message UM protocol does not explicitly authenticate the ephemeral DH values exchanged by the parties nor it includes them under the KDF. Had these ephemeral values been included under the KDF computation then one could prove the basic security of the protocol against known key attacks. Indeed, such a proof has been presented in [6] under the random oracle model (namely, where the hash function used to implement the KDF is modeled as a random function) assuming that parties provide proofs of possession for their static private keys[1]. Another way to make the protocol secure (in the above sense) is to require the performance of key confirmation steps as long as the ephemeral DH values are included in the information authenticated by these steps (but these steps are omitted anyway in the core 2-message protocol).

We stress that even if the UM protocol is modified in one of the above ways to make it secure it still seems inferior in performance and security properties than the other protocol specified in 800-56, namely the MQV protocol (or its more secure variant discussed below).

2.2 MQV

Due to the obvious weaknesses of the UM protocol we are left with the MQV protocol as the main key agreement protocol in the standard. Following [14, 11], the MQV protocol defines the shared secret as the value $g^{(ad+x)(be+y)}$ where d, e are values (each of length |q|/2) computed as the truncated numerical representation of the ephemeral DH values g^x, g^y. The MQV protocol is very attractive due to its performance and claimed security properties. Yet, in spite of its popularity and attractiveness this protocol eluded formal analysis until very recently. In a recent work [10], however, this author shows that MQV

[1] It may be the case that by including the parties' identities under the key derivation function the need for proofs of possession can be eliminated but this requires further analysis.

falls short of delivering provable security. It is shown that in the formal model of [4] the protocol fails to attacks that rule out the possibility of proving the protocol secure in this model. Fortunately, these shortcomings of MQV can be avoided with a simple variant of the protocol, named HMQV, that [10] shows to provably satisfy all the stated security goals of MQV at the same, or even less, complexity than the original MQV.

The main difference between HMQV and MQV is in the computation of the shared secret or, more specifically, the computation of the values d and e used to derive the shared secret. We refer to [10] for the exact details. Roughly speaking, in HMQV we have d = H(gx, IDb) and e = H(gy, IDa) where IDa and IDb are the identities of the peers to the exchange and H is a hash function modeled as a random oracle. With this modification all the security properties claimed for MQV can be formally proven and, moreover, some of the steps taken by MQV can be avoided thus resulting in less complexity and improved performance in some cases. Very importantly, HMQV does not require proofs of possession of private keys (as we show below in Section 3.1 specifying proofs of possession in a sound and secure way is hard). Also, the need for ephemeral (and static) public key validation is essential only when performing the one-pass MQV protocol. In other cases (the 2- and 3-message variants), this validation is needed only if one wants to protect the protocol against the disclosure of ephemeral exponents (something we consider important in general but not necessary if ephemeral exponents are as protected as static private keys – as an analogy such a protection must be provided for systems that implement the DSA signature algorithm).

3 Additional Mechanisms

3.1 Proof of Possession (PoP)

The specification in 800-56 mandates that each participant in a key establishment protocol receives assurance of the fact that the peer to the exchange has been (currently or at some point in the past) in possession of the private key corresponding to the peer's static public key. Such a proof by the owner of a public key is usually referred to as a "proof of possession" (PoP). As we see here, not only PoPs do not help in making the specified protocols secure but they add both complexity and further security vulnerabilities.

800-56 proposes different ways to accomplish PoPs. At a minimum each party is required to provide a PoP to the CA as a requisite for receiving a certificate for its public key. In this way, by seeing a certificate of Alice's public key, Bob can be sure that Alice was in possession of the private key at the time of public key registration. 800-56 further recommends that parties seek more fresh assurance by running more frequent PoP protocols with the owner of a public key.

PoPs present several difficult issues at the level of system complexity, performance, implementation and security. Two main issues are: (1) the need to rely on the CA in doing these checks or, alternatively, to go through the expensive operation of providing these proofs in real time to the exchange's peer; (2) implementing the PoP operation in a way that provides the required assurance without compromising the normal use of the key. The first puts the burden of implementing and mandating the PoP on the CA; yet, in

many real-life scenarios CAs (or other certificate issuing mechanisms) do not implement these measures. The second issue is more delicate and very hard to get it right; indeed, the specification in 800-56 is a good example of this difficulty. Let us elaborate.

The document suggests that when Alice is to provide Bob with a PoP then Alice and Bob run a key establishment (KE) protocol using Alice's private/public key pair. It is argued that if the protocol includes key confirmation then the successful completion of the protocol will show Bob that Alice knows her private key. Now, this presents a circular problem: supposedly, the reason to request PoP in the first place is to provide some assurance necessary for the security of the KE protocol. But now we are using the very same KE protocol, that required the PoP for its security, as the means to provide the PoP (in particular, this first run of the KE protocol with the given private key cannot be guaranteed to be secure since it was not preceded by a PoP).

Moreover, to further illustrate the failure of such a measure (running a KE as a proof of possession), consider a KE protocol such as the UM (the insecure version in 800-56 or the more secure version described in Section 2.1). All that is needed to run a successful execution of this protocol is knowledge of the static shared key $Z_s = g^{ab}$ where a and b are the private keys of Alice and Bob. Therefore, an attacker that learns Z_s can "prove knowledge" of a according to 800-56 even though the attacker may have never learn a. In particular, any party knowing Bob's key b can prove to Bob possession of the private key of any other party! In other protocols, such as MQV, providing PoP by running the protocol can also be insufficient since there is always an ephemeral value whose knowledge suffices to successfully complete the protocol.

An additional vulnerability created by the use of a KE protocol to provide PoP is that the private key may be created for exclusive use by some specific applications in some special settings. Hence, having to use the key in an exchange with, say, the CA would contradict the intended usage of the key and jeopardize its security. The best solution to the PoP problem is to avoid these proofs all together through the use of protocols that do not require them for security. (In theory, ZK proofs could help in providing sound PoPs but such techniques are seldom implemented – not even discussed in 800-56 – and even if they are implemented the need to expose the key to the PoP interface also presents a vulnerability.)

The bottom line is that getting PoPs right is very hard, and they add non-trivial complexity to a security system. Moreover, not in all cases this measure will be implemented (especially when it is not integral part of the KE protocol) and, even when implemented PoPs may add a security vulnerability. Hence, one should recommend to avoid PoPs except when absolutely necessary. Is this the case of 800-56? We do not know if it helps in any way to the original MQV, but we do know that it does not help the UKS attack of Kaliski. For HMQV the situation is far better: we have a proof that the protocol is secure without PoPs; hence, by replacing MQV with HMQV in the spec of 800-56 the PoP issue is completely avoided. As for the UM protocol as defined in 800-56 the protocol is insecure with and without PoPs. If the protocol is re-defined as recommended in Section 2.1 then PoPs may or may not be needed. They are definitely

needed when the identities are not included under the KDF, and KC is not performed. However, since the spec mandates identities under the KDF one may be able to show that PoP is not necessary in this case. This, however, requires further analysis.

3.2 Key Derivation Function (KDF)

As said in Section 2 all protocols specified in 800-56 generate a shared secret (denoted Z) between the peers to the exchange from which further keying material is derived. The function used for this further key derivation is called a key derivation function (KDF). The standard defines a default KDF and allows for two other variants (for use with the TLS and IKEv2 protocols only). In all cases the KDF uses a hash function H which is applied to the shared secret Z, to the identities of the peers, and in particular cases to one or two nonces. The default KDF uses a "counter mode" style derivation to produce the required keying material.

While the general concept of KDF is correct, we believe that the default KDF in 800-56 could be improved. The approach we recommend follows the design of the KDF in IKEv2 [8], namely, to define the KDF process in two steps (for some detailed rationale see [9, 5]). In the first, a key K of a given size is derived by hashing the shared secret (possibly with additional information such as the parties' identities); in the second step, K is used as the key to a PRF (pseudorandom function) which is then repeatedly used to derive as much keying material as needed. The advantage of this approach is its modularity and analytical soundness. In particular, the second stage dispenses completely of the need for random oracles and related idealized primitives. Such primitive, when needed (as in the case of the UM and HMQV protocols), can be confined to the first step. In some cases one may use the same PRF family for the derivation of K also in the first step (such is the case of IKEv2). A PRF can be implemented using a variety of methods, such as using the HMAC algorithm or using a block cipher in CBC-MAC mode.

In addition, and regardless of whether one goes for the two-step approach above or the one-step of the current specification, we recommend using a "feedback mode" rather than "counter mode" in the derivation of keying material. Such feedback mode is used in IKEv2 [8] (and explained in Appendix B of 800-56). Its main advantage over counter mode is that successive blocks of keying material are derived from computationally independent inputs to the hash or PRF functions (in counter mode such inputs differ by very few bits).

Note: In the implementation of the counter-mode KDF in 800-56 the counter is concatenated before the value of Z. We are not sure about the reason for this definition but we point out that Preneel and van Oorschot [15] showed that in some cases spreading the key over two input blocks of a hash function may weaken the scheme. This may be the case, in principle, when prepending the counter as in 800-56.

3.3 Key Confirmation

The protocols specified in 800-56 accommodate the possible (optional) performance of explicit key confirmation (KC) by which a party in the protocol proves to its peer

knowledge of the shared secret (this is done by transmitting a MAC value computed using a freshly derived key from the shared secret). Providing KC may be significant for operational reasons in some cases; e.g., an application may need some assurance that the keying material established through a KE protocol has been correctly calculated before using these keys. More significantly, as pointed out in [10], in the case of implicitly authenticated DH protocols the execution of KC is necessary to ensure the full property of perfect forward secrecy (PFS) (this should be pointed out in 800-56).

The specific mechanisms defined in 800-56 for key confirmation seem correct (they may be simplified with a protocol such as HMQV). Our only (strong) criticism in this context is the way that 800-56 uses the KC mechanism in the context of providing a proof of possession of private keys, an issue discussed in Section 3.1.

3.4 Public key validation

One of the important elements embedded in the 800-56 specifications is the need for validation of public keys, both static and ephemeral (the latter refers to ephemeral DH values). This validation includes checking that a given DH value lies in the right group and has the right order. The cost of such a validation may be negligible for static public keys (if done at certification) but may be substantial for ephemeral DH values (in which case, the validation needs to be performed with each run of the KE protocol).

Specifically, 800-56 considers two main tests for ephemeral DH values: membership in a supergroup (such as an elliptic curve or Z_p) and membership in a prime-order subgroup (i.e., membership in a group generated by the specified generator). The first test is inexpensive but the latter may be costly depending on the underlying group. More precisely, let's consider the two group families from 800-56: elliptic curves and finite fields, and let's denote by N the order of the corresponding super group (i.e., N is the order of the containing curve in the ECC case while N = p - 1 in the Z_p case), and by q the prime order of the sub-group generated by a given element g. Testing that a DH value X is of order q can be done by testing that either (i) xq = 1 or that (ii) Xh 6= 1 where h is the co-factor defined as h = N/q. The first option costs a full exponentiation (i.e., it uses a |q|-bit exponent) while the latter uses an exponent of size |h|. For the parameters specified in 800-56 for the FFC case h is much larger than q while for the ECC case h is small (|h| _ 32). Therefore, it makes sense to use test (i) in the FFC case and (ii) for ECC. What 800-56 specifies is that (i) be performed in the FFC case. For ECC the test is made optional; instead, the computed shared secret is ensured to be of prime order by exponentiating this value to the power of h (this has the same computational cost than an explicit membership test).

These measures seem to be intended as a protection against some known attacks such as small-group and Lim-Lee attacks [12]. Unfortunately, lacking a formal analysis of these protocols it is hard to know to what extent these measures are needed and, more significantly, to what extent they achieve their intended goals. In particular, is the co-factor exponentiation in the ECC case the right replacement for a membership test of ephemeral DH values? In the the FFC case, where membership tests are expensive, are they always needed?

We do not know the exact answers for MQV and UM. In contrast, in the case of HMQV we have an exact characterization for the need of membership validation. The analysis from [10] shows that these tests are not needed for the basic security of the 2- and 3-message variants of the protocol but are essential for the security of the one-pass protocol. If one is interested to protect the secrecy of static private keys in the case that the ephemeral private DH exponents are revealed then the test is necessary also in the interactive cases. This establishes a security-performance trade-off depending on the protocols used and the level of protection provided to ephemeral exponents. In particular, if one uses the interactive modes of the protocol (2 or 3 messages) and the ephemeral exponents are protected as well as the long-term static private key (as is assumed in some implementations, such as those of the DSS signature scheme) then the ephemeral validation can be omitted. We note that with the parameters considered in 800-56 this trade-off is significant in the FFC case; for ECC groups, the specification that h be small makes the membership test of small computational cost.

If HMQV is adopted the co-factor exponentiation can be completely avoided thus making the FFC and ECC versions of the protocol more uniform. The only difference is in the way the membership test is performed (through q or h exponentiation).

3.5 Temporal ordering of static and ephemeral keys

In Section 5.6.4.3 of 800-56 (page 41) it is required that a party make sure that the generation of the peer's static public key preceded the creation (or transmission) of the party's own ephemeral DH value. No rationale is provided for this "obscure" and cumbersome requirement (naive as it looks this requirement may be non-trivial to accommodate; for example, it may disallow for simultaneous transmission of static and ephemeral public keys thus incurring in extra protocol flows).

We speculate that it may be intended to prevent Kaliski's UKS attack against MQV, probably in combination with the requirement for proofs of possession for static private keys and the inclusion of identities under the KDF. But is this required? And does it really prevent UKS attacks? This seems as a perfect example for "safety margins" in the design intended to compensate for the lack of clear analysis of a protocol, which comes at the expense of additional complexity and performance degradation.

Fortunately, with the available analysis of HMQV [10] we know that we can safely dispense of such complex measures while still having a guarantee of security against UKS and related authentication attacks.

3.6 Miscellaneous

1. The definition of cryptographic primitives in 800-56, even if intentionally informal, should be more accurate. In particular, MAC functions are not just one-way functions (Sec 5.2 of [1]). They are keyed families with the property of being unforgeable under a chosen message attack. Similarly, for PRFs the property of collision resistance is irrelevant (page 115 of [1]); instead, they are keyed families with unpredictable

outputs. In general, the use of hash functions in this document is not as collision resistant functions (page 14 of [1]) but rather as generating random outputs.

2. 160 bits for the subgroup order may well prove insufficient even for 80-bit security requirements; 800-56 should recommend larger parameters (even with 80-bit security in mind).

3. If there is some good reason to standardize on C(2, 0) protocols (I doubt it is a good idea) then at least strongly caution about the difficulty of making such a protocol secure even in combination with digital signatures. The current text (page 69) is a perfect receipt for home-grown wrong protocols.

4. In defining C(0, 2) protocols you should warn about the possibility of nonce replays and the adversarial effect of such a replay especially if the uniqueness and freshness of the nonce cannot be verified by the recipient.

5. In Section 5.6.3.1 of [1] several ways of checking that one possesses the right private key for a given public key are proposed. Considering that the whole document is about DH keys why not just check that the given pair (x,X) of private-public keys satisfies X = gx? Is this what it is meant in option 3? Why are other methods needed (is it for the case that the private key is not available for a calculation as above even to the owner of the key?). Please explain. Also, as in the case of PoPs (Section 3.1), validating or testing a key via a KE with the CA should be discouraged; in particular since an application's policy may strictly enforce the use of the private key in specific environments (not for interaction with the CA).

4 Concluding Remarks and Recommendations

We conclude by discussing the essential role of formal analysis in the design and choice of cryptographic mechanisms, and by providing some specific recommendations.

4.1 The fundamental role of analysis

The security analysis of protocols not only gives us a significant assurance of security (as long as the protocol is implemented with secure primitives) but also a precise understanding of the role and rationale of each design element. The recent results leading to the design and analysis of the HMQV protocol [10] serve to highlight this role of analysis. Examples include the essential need to bind the key computation to identities (lacking in the original MQV protocol and the reason for the vulnerability of MQV to UKS attacks), the need for hashing the shared secret (stated as non-essential in [11]), the dispensability of proofs of possession, what can and cannot be stored in temporary storage, when and for what s PK validation required, and the exact added value of key confirmation. (The latter is required not just to provide the operational certainty that the peer computed the correct key but to ensure full PFS – on the other hand the usual justification of KC for preventing UKS is not necessarily true if ephemeral information leaks).

Hence, while security, formal or otherwise, is never an absolute term but rather a statement relative to a specific adversarial model and execution environment, a sound analysis provides us with well-defined properties and statements amenable to the examination and review by others, as well as with a guide for making informed decisions about the functionality and necessity of each protocol component, and for understanding the interaction between these components. It allows the designer (and analyst) to discern between the essential, the desirable and the dispensable elements of a design. The result is not just a more secure scheme but also one with less complexity and better performance.

At a time when we demand the best (almost perfect) security from basic encryption and hash functions, and having witnessed the way in which initially-mild attacks shaked our confidence in such functions, we can only hope that the applied-cryptography community and its representing standard bodies will see formal analysis as a requirement, and main source of confidence, when adopting protocols for wide use. These analyses can (and must) be verified by the community at large (in contrast, ad-hoc designs do not even provide the "luxury" of judging well-defined security properties). This is all the more significant in the case of influential standards such as NIST's that serve not only to regulate use of cryptography by government agencies but to provide guidance and standards for the industry at large, and whose mandated mechanisms are likely to be adopted by other organizations.

4.2 Recommendations

In this section we summarize some of our specific recommendations. The central recommendation is to choose the core key agreement protocols in the standard from the set of existing protocols for which security is well understood and proven in a sound formal model. The exact choices may depend on what the engineering requirements are and the preferred security properties (unfortunately these are not explained in 800-56). Assuming that NIST will be interested in protocols which will require minimal changes to the current spec then the obvious choices would be to (1) correct the UM protocol via the inclusion of the ephemeral DH values under the KDF computation; and (2) replace the current MQV specification with the similar but analytically superior HMQV protocol. The latter requires little change to the current protocol: just replace the truncated DH values used as exponents in the computation of the shared secret with a value computed as the hash of the party's DH value and peer's identity (see Section 2.2). Also, make the ECC and FFC versions of the protocol identical except for the way prime-order tests (when required) are performed (see Section 3.4). In particular, no need for the difference arising from the co-factor involvement in the computation of the shared secret.

In the case of HMQV, remove any requirement for proofs of possession. These may be left for the corrected UM protocol at least until clearer results on the need for these proofs are obtained (as said earlier, without including the identities under the KDF such a PoP is essential but it may be otherwise unnecessary). If protocols that need PoPs will still be specified in the document then better guidelines for the way to perform these proofs must be provided, including warnings about the dangers of these mechanisms as we pointed

out in Section 3.1. Eliminate the requirement to check that static private keys were generated before ephemeral DH values.

Replace the preferred mechanism for KDF with the one described in Section 3.2, namely, using a two-step computation involving a PRF in feedback mode. The current mechanisms can be allowed as well. Take care of the additional comments in Section 3.6. Also important would be to provide some guidance as for the level of protection that different secret elements in the protocol require: from the obvious need to protect the static private key as the most valuable secret in the protocol to the specification of which computations should be computed in secure hardware (e.g., not just the shared secret but its hashing too) to the specification of values that affect a single session but whose disclosure do not compromise other sessions (such as the session key itself and ephemeral exponents if membership tests are performed). Similarly, indicate whether can one cache the Zs values used in UM or whether these values must be generated with each run of the protocol.

Finally, the document must provide rationale for its choices, ranging from basic operational and engineering considerations to the theoretical backing of the specified mechanisms. At the very least, it should provide pointers to other documents and works where this rationale is given. In addition, to aid implementations and analysis, the spec should be streamlined with a uniform and systematic description of protocols for both ECC and FFC cases except, of course, for the difference in the underlying algebra (but as protocols they should be essentially the same). In addition, the document should provide similar names to the ECC and FFC variants (if differentiation is required call the protocols ECC-UM and FFC-UM rather than “Full UM” and “dhHybrid1” as currently named). Yet another way to further simplify the spec is to minimize the number of mechanisms described: is the description of both FFC DH and ECC CDH really needed? Are all the C(·, ·) variants needed too? And what is the added value of having both UM and MQV (or HMQV), how should one choose one or the other in an application?

References

[1] “Draft Special Publication 800-56, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”, July 2005. Available from http://csrc.nist.gov/publications/drafts.html

[2] American National Standard (ANSI) X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[3] S. Blake-Wilson, D. Johnson and A. Menezes, “Key exchange protocols and their security analysis,” Sixth IMA International Conference on Cryptography and Coding, 1997.

[4] Canetti, R., and Krawczyk, H., “Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels”, Eurocrypt’2001, LNCS Vol. 2045. Full version in: Cryptology ePrint Archive (http://eprint.iacr.org/), Report 2001/040.

[5] Dodis, Y., Gennaro, R., H°astad, J., Krawczyk H., and Rabin, T., “Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes”, Crypto’04, LNCS 3152, pp. 494–510.

[6] Ik Rae Jeong, Jonathan Katz, Dong Hoon Lee, "One-Round Protocols for Two-Party Authenticated Key Exchange", ACNS 2004: 220-232
[7] B. Kaliski, "An unknown key-share attack on the MQV key agreement protocol", ACM Transactions on Information and System Security (TISSEC). Vol. 4 No. 3, 2001, pp. 275–288.
[8] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol", draft-ietfipsec-ikev2-xx.txt, 2004 (to be published as an RFC).
[9] H. Krawczyk, "SIGMA: The 'SiGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols", Crypto '03, LNCS No. 2729, pp. 400–425, 2003.
[10] H. Krawczyk, "HMQV: A High-Performance Secure Diffie-Hellman Protocol", Crypto'05. Full version: http://eprint.iacr.org/2005/176
[11] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient Protocol for Authenticated Key Agreement", Designs, Codes and Cryptography, 28, 119-134, 2003.
[12] C. H. Lim and P.J. Lee, "A Key Recovery Attack on Discrete Logbased Schemes Using a Prime Order Subgroup", Advances in Cryptology – CRYPTO 97 Proceedings, Lecture Notes in Computer Science, Springer-Verlag Vol. 1294, B. Kaliski, ed, 1997, pp. 249–263
[13] T. Matsumoto, Y. Takashima, and H. Imai, "On seeking smart publickey distribution systems", Trans. IECE of Japan, 1986, E69(2), pp. 99-106.
[14] A. Menezes, M. Qu, and S. Vanstone, "Some new key agreement protocols providing mutual implicit authentication", Second Workshop on Selected Areas in Cryptography (SAC 95), 1995.
[15] B. Preneel and P. van Oorschot, "MD-x MAC and building fast MACs from hash functions," Crypto'95, LNCS 963.

From: Ford Long Wong <fw242@cam.ac.uk>
Date: 12 Aug 2005 13:39:24 +0100

Typos:

Pg 70, Figure 6, right column, 1. line 2 - "...and U's static static and..." should be "...and U's static and..."

Pg 105, Section 8.4.8, Para 2, line 2 - "EphemData_U = Nonce_U and EphemData_V = Nonce_V:" should be "EphemData_U = Null and EphemData_V = Nonce_V:"
Para 4, line 1 - "MacDataU = "KC_1_U" || ID_U || ID_V || Nonce_U || Nonce_V || {|| Text_1}" should be "MacData_U = "KC_1_U" || ID_U || ID_V || EphemData_U || Nonce_V || {|| Text_1}" and "MacData_U" should be italicized.

Others:

Pg 92, Section 6.3.3.
Here is a distinct and unique problem for the C(0,2) scheme, because it is the only scheme that uses no ephemeral key pairs, and thus the following sentence could perhaps(?) be considered for inclusion at the end of Para 2 - "... But if the shared secret of two parties is compromised, even if neither party's static private key is compromised, then all future derived secret keying material involving the two parties can be compromised." By looking at what is covered in the similar Sections of 6.1.1.5, 6.1.2.3, 6.2.1.5 and 6.2.2.3, mentioning this problem seems consistent. While it can be argued that if the recommendations in Section 6.3.2 are followed, then the results of the intermediate calculation such as the shared secret should have been destroyed and not available to an adversary, however, other "Rationale" sections, eg. 6.2.2.3 themselves sketch out security risks if ephemeral private keys (which should have been destroyed after use as recommended) are compromised.

Pg 95, Section 8.1, Para 3 - "... The benefits of assurance of current possession of a static private key incorporate the benefits of assurance of prior possession, but not vice versa... " This makes an assertion which seems somewhat debatable to me.

Pg 29, Section 5.5, and Pg 110, Appendix A, Para 5. The recommendation requires the static-key domain parameters and the ephemeral-key domain parameters to be the same for any given run of a scheme. I can see that it's more convenient and less complex that way.

On the other hand, I can also envisage scenarios where as a user I would require longer static keys for long-term use, which would be rarely certified, but which an adversary may attempt to break over a long time, and yet I would require only shorter ephemeral keys because I can re-generate these all the time. So perhaps this recommendation could be more flexible...

- Ford

Date: Fri, 19 Aug 2005 14:47:52 -0400
From: Russ Housley <housley@vigilsec.com>

I am writing to make you aware of a compatibility problem between RFC 2631 and RFC 3278 with the Draft Special Publication 800-56, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography.

The draft SP 800-56 requires some inputs to the KDF that are optional in X9.42 and X9.63.

RFC 2631 is a profile of X9.42, and it does not include the identities of the key holders as inputs to the key derivation function (KDF). This is especially important in the case of ephemeral-static Diffie-Hellman, as the originator does not have a certificate. Without a certificate, it is ambiguous what identity might be included.

RFC 3278 is a profile of X9.63, and it does not includes the identities of the key holders as inputs to the KDF.

It is important that implementations that conform to RFC 2631 and RFC 3278 are able to be evaluated against FIPS 140, and it seems that this requirement will prevent implementations that conform to these RFCs from being certified under FIPS 140.

The S/MIME environment is not the only one where including identities will lead to deployment problems. Mandatory inclusion of identities is problematic in applications where one or more of the parties wishes to remain anonymous, and it is problematic for SSL/TLS clients that have no natural identity.

Please make the inclusion of identities of the key holders optional inputs to the KDF. This could be accomplished by changing the "shall" to "should" in the discussion of PartyUInfo and PartyVInfo.

Thanks for your attention,
Russ

From Joshua E. Hill
InfoGard

- Section 4.2, first paragraph: The section "Note that in a given transaction, one or more of the indicated actions may be omitted (depending on the key agreement scheme). For example, key confirmation may be omitted, depending on the security requirements of the participants." It is unclear if this has the effect of allowing the vendor/implementer to remove arbitrary portions of the described establishment mechanisms. We suspect that the intent of these sentences is to indicate that certain actions are optional, and that the optional portions are identified later in the document, but this should be made more clear.

- The first sentence of Section 4.3 seems to have the effect of making a process described in a figure as binding. It would be better to make the "shall" statement pertain to text, and refer to the figure for clarity.

- Section 5.2, second paragraph: What does the sentence "The MAC algorithm [...] shall be used to validate implementations of the key establishment schemes specified in this Recommendation." mean? In what way is the MAC algorithm used to "validate implementations of the key establishment schemes"?

- What does Section 5.2.3 require? What is an "Implementation Validation Message"?

- In Section 5.2, the footnote seems superfluous, given that this is explicitly mentioned in bullet 2a.

- In Section 5.3, why was the 80 bit level of security not included?

- In Section 5.4, what does the term "negligible chance" mean in this context? What is a reasonable value for "negligible"?

- Section 5.4, bullet #1: What type of "entropy" is intended?

- Section 5.4, bullet #4: Why is this option separately discussed? This appears to be a combination of #2 and #3, and the sentence just prior to the list explicitly allows for nonce construction using more than one of these options.

- Section 5.5.1.1, table #1: Option FC is not consistent with any level of security noted in NIST SP800-57, Section 5.6.1 Table 2. It appears that the intent of FC is to provide roughly 128 bits of security, which would (according to SP800-57) require 'p' to be 3072 bits long, not 2048 bits long.

- In Section 5.5.1.1, the last sentence of the second paragraph refers to Section 5.5.1.2, but it isn't clear why. (5.5.1.2 isn't a section in SP800-57, so we assume that the reference was to SP800-56, which doesn't seem applicable.)

- Section 5.5.2, bullet 1: There is an extra space in "domain parameter".

- Section 5.6.1.1 makes reference to domain parameter generation as documented in FIPS 186-3. A draft of FIPS 186-3 has not been released for public review, so it is not possible for reviewers to comment on the appropriateness or applicability of this reference. (FIPS 186-3 is also referenced in 5.5.1.1, 5.5.1.2, 5.5.2, 5.6.1.2, and Appendix A.)

- Sections 5.6.2.1, 5.6.2.2, 5.6.2.3, etc: Requirements are phrased similarly to "The owner shall know the method(s) of...". It is unclear how the CMVP testing laboratories are going to test the owners for knowledge of anything. The requirements should be phrased in terms of vendor documentation provided, or requirements for devices that implement portions of the recommendations, information present in messages, etc.

- Section 5.6.2.1, bullets 3 and 4: There is reference to Section 5.6.1, but it isn't clear that this is the intended reference. In each case, is there a more specific subsection being referenced?

- Section 5.6.3.1: The use of the term "owner" twice seems to imply that the referenced owners are separate entities. If this is not the intent of the requirement, it should be rephrased.

- Section 5.6.3.1: As the owner likely possesses both the public and private keys, couldn't they perform both sides of an exchange with themselves (using a separate key), and verify that both "sides" received the same shared secret? This is similar to option 1, except that no key confirmation is necessary (as the shared secret for both "sides" can be directly compared).

- Section 5.6.4.2, in bullet 4: There is a parenthetical example that appears to be a normative requirement.

- In Section 5.6.4.3 (and 5.8.4, 6.1.1.1, 6.1.1.2, 6.1.1.3, etc), the term "destroyed" is used without mention of zeroization. In other references to "destroy" the parenthetical "(i.e., zeroized)" is present. Perhaps the most reasonable way of proceeding is to add “destroy” to the definition area, and indicate that the only acceptable method of destruction is zeroization.

- In general, there are many places (Sections 5.7, 6.1.1.1, 6.1.1.2, 6.1.1.3, 6.1.1.4, 6.1.2.1, 6.2.1.2, etc.) where descriptions of establishment schemes are presented as "shall" requirements, which seems a bit odd. The document is (in some settings) normative in its descriptions without the use of the “descriptive shall”. A particular key establishment scheme either implements one of the described key establishment methods, or it does not. There doesn't seem to be much point in making a description a "shall" statement.

- In response to the last comment set that requested that the establishment schemes used in TLS and IPSec be allowed, it appears that an allowance was made for the KDF used in TLS and IKEv2. We have a number of comments regarding this approach:
    - For IPSec, the KDF used within IKEv2 was allowed for use, but not the KDF used in IKEv1. IKEv2 is not approved as an RFC at this point, and IKEv1 is dominant in the market, and is in widespread use throughout many federal institutions. If IKEv1 was excluded for security reasons, then the extreme disruption caused by excluding IKEv1 as an Approved key establishment scheme may be justifiable. If this is not the case, then I advice you to adopt IKEv1 in addition to IKEv2.
    - The inclusion of the TLS and IKEv2 KDF is a good first step, but it still isn't clear if any of the modes of these two protocols are actually allowed, or just their KDF. It would be very helpful if an informative appendix was added that sketched the various relevant modes of use of IKE (v1 and v2) and TLS, and outlined which could be made compliant at the protocol level with the requirements of this recommendation, and what additional requirements must be separately fulfilled.
    - IKE (in general, irrespective of version) may have a few issues. It seems that IKE would be an instance of C(2, 0, FFC DH) in its common FFC modes (Groups 2, 5, 14, 15 seem relevant given the parameter size sets discussed in SP800-56 Section 5.5.1.1). IKE has a set of standardized domain parameters that are integrated into the standardized key exchange (p, and g in the parlance of SP800-56). It is unclear if SP800-56 would allow use of these parameters, as they were likely not generated as per FIPS 186-3 or ANSI X9.42. For example, in Group 2 the p value is a 1024 bit prime. The factors of (p-1) are 2 and a large (approximately 1023 bits) prime. We may be able to designate this 1023 bit prime as 'q' (which appears to be acceptable). For this group, g=2. For these values, $g^q \bmod p = 1$, so this selection of p may be valid, even if it was not selected as described in ANSI X9.42 Appendix B.2. What groups might be acceptable?
    - It appears that the TLS DHE cipher suites may be allowed as a C(2, 0, FFC DH), but the TLS DH cipher suites which would be categorized as a C(1, 1, FFC DH) system, and thus require that the client/initiator obtains assurance that the server/responder is in possession of the appropriate static private key as per Section 5.6.3.2 (which in turn allows the methods in Sections 5.6.3.2.1 and 5.6.3.2.2). Section 5.6.3.2.1 allows for explicit key confirmation (as described in Section 8 of the document), but unfortunately this method appears to be incompatible with the TLS protocol. Section 5.6.3.2.2 requires that the DH private key proof of possession was issued by a CA, which is not currently a common CA service.

- The specification of most of the establishment methods includes use of the apparent literal strings "Invalid" and "Failure". The document should be clarified to indicate that using these particular literal strings is not required, as long as the correct error status information is conveyed.

- In Section 5.8.1, in the Process section, the 32 bit "counter" value is described as "big-endian". In Section 5.8.2, the Input section, Step 3.1.2 the analogous counter byte ordering is not described.

- In Section 5.8.3, the IKEv2 PRF should not be used to produce more than 255 blocks of output (the counter used is a 1 byte value). This block number restriction is mentioned in B.4, but not in Section 5.8.3.

- In Section 5.8.4, it would be useful to indicate that the master secret is considered to be "keying material" because it is derived from the shared secret of a key establishment scheme and is a value that is intended to be retained in order to facilitate TLS session resumes. Some confusion on this point seems possible, as the master secret is ultimately input into the PRF (with the session nonces) to produce the keys used for the TLS session.

- Sections 6.1.1.1, 6.1.1.2, etc., actions bullet 5: The use of the concatenation operator "||" in italics is confusing as it does not look sufficiently like the normal '||' operator.

- In Sections 6.1.1.3, 6.1.1.4, actions bullet 1 references Section 5.6.5, which is not a valid section in this document.

- Section 6.1.1.4, actions bullet 3: References 5.7.2.3, but the reference should probably be more specific and reference 5.7.2.3.1.

- In Section 6.2.1.3, the first paragraph makes reference to Section 6.1.1. This reference should probably be 6.2.1.

- Section 6.2.1.4, actions bullet 2: The reference to Section 5.7.2.3 should be more specific, likely 5.7.2.3.2.

- In Section 8.4, the last sentence likely ought to be a "shall" requirement, rather than a "should" requirement (it ought to be 'shall' at least in the cases where key confirmation is being used to provide required proof of possession). If this requirement remains a 'should', then there isn't much point in requiring that a key confirmation occur at any point.

- In Section 10, the implementer/vendor should also identify the method of generation of domain parameters in the case where they are generating them as per ANSI X9.62 (as allowed in 5.5.1.2).

- In Section 10, the final bulleted list, fifth bullet from the bottom, "The key schemes available" should probably be "The key establishment schemes available".

Date: Fri, 19 Aug 2005 13:47:26 -0700
From: Brian LaMacchia <bal@exchange.microsoft.com>
To: <ebarker@nist.gov>
CC: "Kristin Lauter" <klauter@microsoft.com>

We are pleased to submit the following comments on the July 2005 Draft of SP 800-56 on behalf of Microsoft Corporation, Redmond, WA.

1. Choice of Algorithms Included in SP 800-56

Our first comment concerns the set of algorithms NIST has chosen to include within SP 800-56. Generally, for each "category" of key establishment, SP 800-56 provides two distinct protocols: one based on Diffie-Hellman and one based on MQV. Further, such choices are provided for both Finite Field Cryptography ("FFC") and Elliptic Curve Cryptography ("ECC"), so for some categories (such as C(2,2)) there are four possible algorithms from which implementers can choose to be compliant with SP 800-56.

If the goal of SP 800-56 is to guarantee interoperability between compliant parties, then we believe SP 800-56 currently provides too many algorithm choices. Based on past experience in Internet standardization efforts, there needs to be a single approved, mandatory-to-implement algorithm in order to guarantee widespread interoperability. If this is NIST's goal with SP 800-56 then we think NIST needs to choose one algorithm (or one algorithm family) as the mandatory-to-implement algorithm for SP 800-56. It is acceptable to specify multiple functionally-equivalent algorithms, but there needs to be exactly one algorithm that compliant implementations MUST implement (to use the language of IETF RFC 2119 [Br97]); other functionally-equivalent algorithms SHOULD or MAY be implemented but are not required.

On the other hand, if the goal of SP 800-56 is to define a broad pool of approved algorithms from which an individual algorithm may be chosen as mandatory-toimplement for a particular protocol (e.g. TLS), then we believe that the current pool of algorithms specified in SP 800-56 is too small. Indeed, given the amount of active research on provable security for such algorithms and the set of MQV variants that have been proposed within the last few months [Kr05, Me05], we believe it would be imprudent to fix the pool of approved algorithms at this time. Instead, if defining a broad pool of key establishment schemes is SP 800-56's primary goal, then we recommend that SP 800-56 define a framework for key establishment schemes and a process by which new algorithms may be added to the "SP 800-56 approved pool" over time. Such a mechanism would allow additional, tailored algorithms to be added to the SP 800-56 pool as necessary to meet the needs of particular communication protocols.

To illustrate the benefit of having such a framework and dynamic approval process, we would like to call NIST's attention to some of our recent work on key establishment schemes. First, in "Security Analysis of KEA Authenticated Key Exchange" by Lauter and Mityagin [LM05], available from the IACR ePrint Archive at http://eprint.iacr.org/2005/265, is a definition of the "KEA+" algorithm, which is a key establishment scheme in the spirit of SP 800-56 that is provably secure in the Canetti-

Krawczyk model. Second, in Appendix A of these comments we have included a brief description of two additional algorithms – Naxos1 and Naxos2 – we have developed with Mityagin that we believe are also provably secure in the Canetti-Krawczyk model and have some additional advantages over KEA+. (A paper detailing our security analysis of Naxos1 and Naxos2 is forthcoming.) These algorithms all fit within the general framework of key establishment schemes implied by those algorithms in SP 800-56. Thus, if the goal of SP 800-56 is to establish a pool of qualified algorithms from which a protocol designer should choose, we recommend that SP 800-56 include a process for adding algorithms to the pool and the framework in which those algorithms should operate.

We believe it would be inappropriate to standardize SP 800-56 before deciding whether the goal of SP 800-56 is to provide a pool of approved algorithms or to define an interoperability standard. In the case of the former, we recommend that SP 800-56 be modified to include a well-defined framework for promptly including other algorithms. If the latter is the case, then we believe that there needs to be more discussion and analysis as part of the process of deciding which specific, mandatory-to-implement algorithm will be standardized by SP 800-56.

2. Prohibition against Key Cross Use
Requirement 4 of Section 5.6.4.2 (Specific Requirements on Static Key Pairs) reads as follows:

> A static key pair may be used in more than one key establishment scheme. However, one static public/private key pair **shall not** be used for different purposes (for example, a digital signature key pair **shall not** be used for key establishment or vice versa).

We believe this prohibition against use of a key pair for "different purposes" is overly broad. Specifically, as currently written, this requirement prohibits the use of an ECDH private key as the signing key in an ECDSA-based proof-of-possession protocol. When enrolling for a certificate for a static key establishment key pair, it would be convenient to be able to use the static private key as an ECDSA signing key. We are not aware of any cryptographic reason to prohibit using an ECDH static private key in ECDSA-based proof-of-possession protocols.

Thank you for the opportunity to provide comments on the July 2005 Draft of SP 800-56; please contact us if you would like to discuss any of these points further.

Sincerely,
Brian A. LaMacchia Kristin Lauter
Software Architect, Office of the CTO Researcher, Microsoft Research
bal@microsoft.com klauter@microsoft.com
One Microsoft Way One Microsoft Way
Redmond, WA 98052-6399 Redmond, WA 98052-6399
(425) 703-9906 (phone) (425) 703-8335 (phone)

(425) 936-7329 (fax) (425) 936-7329 (fax)

References

[Br97] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997, available at http://www.ietf.org/rfc/rfc2119.txt.

[Kr05] Hugo Krawczyk, "HMQV: A High-Performance Secure Diffie-Hellman Protocol," available from http://eprint.iacr.org/2005/176

[LM05] Kristin Lauter and Anton Mityagin, "Security Analysis of KEA Authenticated Key Exchange," available at http://eprint.iacr.org/2005/265.

[Me05] Alfred Menezes, "Another look at HMQV," available at http://eprint.iacr.org/2005/205.

Appendix A – The Naxos1 and Naxos2 Authenticated Key Establishment Schemes.

The Naxos1 and Naxos2 protocols are refinements of KEA+ (see [LM05]) which we also believe to be provably secure in the Canetti-Krawczyk model. Additionally, Naxos1 and Naxos2 provide extra protection against revelation of ephemeral secret keys. Naxos1 has the same efficiency as KEA+ or dhHybrid1 in SP 800-56; Naxos2 adds an extra security feature but requires an additional exponentiation to do so.

These new protocols Naxos1 and Naxos2 introduce a cryptographic hash of a static secret key and an ephemeral secret as the new ephemeral exponent on both sides of the exchange. Thus, instead of exchanging g^x and g_y and calculating a shared secret based on g^{bx} and g^{ay}, in Naxos1 and Naxos2 the communicating parties exchange $g^{H(x,a)}$) and $g^{H(y,b)}$ and calculate a shared secret based on the values $g^{bH(x,a)}$ and $g^{aH(y,b)}$. Naxos2 differs from Naxos1 by adding an additional agreed-upon value, $g^{H(x,a)H(y,b)}$), to the derivation of a session key. This additional value ensures that each communicating party possesses the ephemeral secret key.

Below are the descriptions of Naxos1 and Naxos2 following the syntax of the description of KEA+ in [LM05]. These descriptions assume that both parties know each other's certified identities and public keys. All protocols take place in a group G of prime order q with a generator g. The group G can be either a multiplicative subgroup of a finite field or the group of points on an elliptic curve. We write the group operation multiplicatively below. H can be an arbitrary cryptographic hash function.

Note that versions of Naxos1 and Naxos2 with key confirmation are also possible, as in the KEA+ proposal; we can provide details of those protocols if interested.

Naxos1

Initiator	Responder
Identity: $\boldsymbol{ID_A}$ Secret key: $\boldsymbol{a}$ from [$\boldsymbol{1}$…$\boldsymbol{q-1}$] Public key: $\boldsymbol{A} = \boldsymbol{g^a}$ Responder's public key: $\boldsymbol{B}$	Identity: $\boldsymbol{ID_B}$ Secret key: $\boldsymbol{b}$ from [$\boldsymbol{1}$…$\boldsymbol{q-1}$] Public key: $\boldsymbol{B} = \boldsymbol{g^b}$ Initiator's public key: $\boldsymbol{A}$

Session identifier: sid Assumption: Responder's public key is valid (in the group G)	Session identifier: sid Assumption: Initiator's public key is valid (in the group G)
Pick x at random from $[1 \ldots q-1]$ Compute $c = H(x, a)$ Compute $X = g^c$ Send X to the Responder	
	Receive X from Initiator Verify that X is in G; if not, terminate the protocol Pick y at random from $[1 \ldots q-1]$ Compute $d = H(y, b)$ Compute $Y = g^d$ Send Y to Initiator
Receive Y from the Responder Verify that Y is in G; if not, terminate the protocol	
Compute $Z_1 = Y^a$ Compute $Z_2 = B^c$ Compute a session key $K = H(Z_1, Z_2, ID_A, ID_B, sid)$	Compute $Z_1 = A^d$ Compute $Z_2 = X^b$ Compute a session key $K = H(Z_1, Z_2, ID_A, ID_B, sid)$

Naxos2

Initiator	Responder
Identity: ID_A Secret key: a from $[1 \ldots q-1]$ Public key: $A = g^a$ Responder's public key: B Session identifier: sid Assumption: Responder's public key is valid (in the group G)	Identity: ID_B Secret key: b from $[1 \ldots q-1]$ Public key: $B = g^b$ Initiator's public key: A Session identifier: sid Assumption: Initiator's public key is valid (in the group G)
Pick x at random from $[1 \ldots q-1]$ Compute $c = H(x, a)$ Compute $X = g^c$ Send X to the Responder	
	Receive X from Initiator Verify that X is in G; if not, terminate the protocol Pick y at random from $[1 \ldots q-1]$ Compute $d = H(y, b)$ Compute $Y = g^d$ Send Y to Initiator
Receive Y from the Responder Verify that Y is in G; if not, terminate the protocol	

Compute $Z_1 = Y^a$ Compute $Z_2 = B^c$ Compute $Z3 = Y^c$ Compute a session key $K = H(Z_1, Z_2, Z_3, ID_A, ID_B, sid)$	Compute $Z_1 = A^d$ Compute $Z2 = X^b$ Compute $Z_3 = X^d$ Compute a session key $K = H(Z_1, Z_2, Z_3, ID_A, ID_B, sid)$

Date: Fri, 19 Aug 2005 16:35:38 -0700
From: "Robert Jueneman" <rjueneman@spyrus.com>

SPYRUS is pleased to provide comments in response to the July 2005 draft of NIST Special Publication SP 800-56, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography."

First, we would like to thank and congratulate the authors and contributors to SP 800-56 for making the necessary hard choices between a number of conflicting ANSI and other standards. It is an excellent document, very useful, and overall very well written.

We would like to address some of the policy issues, including some that were stated and some that probably need to be stated or clarified.

Partitioning and key confinement issues:

SP 800-56 carefully defines what has to be done for key establishment, but it completely ignores the question of where something is to be done.

We would like to see a delineation that is along the lines of the treatment of such material in ANSI X9.82. To that end, we would suggest the following text:

> In many cases, a complete cryptographic implementation, including key establishment, may be carried out in a system made up of multiple components, not all of which may have the same level of assurance with respect to key confinement. For purposes of reference, a cryptographic system implementation may be divided into seven different nested zones or partitions, as follows:
>
> - **Partition A**: This is the innermost partition, reserved for the state variables and other information associated with the Approved random number generator used for key generation and similar purposes. To the maximum extent possible, these variables should not be accessible by any other function, even if the other function shares the same FIPS 140-2 boundary. For further details, see the discussion of this concept in ANSI X9.82. An exception to this exclusion requirement would be the implementation and storage associated with the use of a Master Key Encryption Key for the device or implementation that is used to encrypt and decrypt keys and other Critical Storage parameters used by or for the outer partitions, including any state variables used by the random number generator. That Master Key Encryption Key should be retained within partition A.
> - **Partition B**: This is the next innermost partition, and is reserved for the cryptographic implementation and storage of personal private keys and persistent symmetric keys.
> - **Partition C**: This is the middle partition, and is reserved for the cryptographic implementation and temporary storage of ephemeral keys, shared secrets, Key Derivation Functions, key confirmation keys, per-message secrets (such as the per-message secret used in the ECDSA signature process), Initialization

Vectors, nonces, etc. On-token hash functions used to perform an atomic hash-and-digitally-sign operation would also be implemented within this partition.

- **Partition D**: This partition is reserved for the cryptographic implementation and temporary storage of Message Encryption Keys, Session Encryption Keys, Volume Encryption Keys, and other operations where performance is a serious issue. It is often implemented in software, making use of a FIPS 140-2 Level 2 implementation within a general-purpose, moderately trusted operating system or application for any specific cryptographic functions.
- **Partition E**: This functionality within this partition is often responsible for certificate parsing and certificate path validation, and for enforcing Enterprise policy regarding who can communicate with whom. Hashing functions that are then sent to a cryptographic token for signature may also be carried out within this partition. Signature validation is also frequently carried out within this partition. It is often implemented in software, making use of a FIPS 140-2 Level 2 implementation within a general-purpose, moderately trusted operating system or application for any specific cryptographic functions.
- **Partition F**: This is the next-to-outermost partition, where the plaintext is created or retrieved and sent to the cryptographic device to be encrypted, and the plaintext resulting from the decryption process is made available or stored. This partition is often involved in receiving and transmitting the ciphertext, depending on whether red/black separation is required. No cryptographic functions are carried out within this partition, but compression functions may be. Perhaps most importantly, the choice of what operations are to be performed, and the specification of what recipients are to receive a given message, is normally performed within this outmost partition.
- **Partition G:** The previous partitions are all considered "Red," in that they are involved in handling the plaintext and/or cryptographic materials. Partition G is the outermost, "Black" partition, generally consisting of those areas of the overall information handling system to which unauthorized users have unrestricted access, e.g., the Internet. All sensitive information should have been encrypted prior to being exposed to partition G, and all information subject to undetected alteration should be digitally signed or cryptographically authenticated.

Optimization Techniques:

The use of optimization techniques for ECC, AES, and SHA-2 should be encouraged. However, no optimization of any kind (e.g., the computation of a per-key table used to speed up point multiplication) shall be performed outside the FIPS 140-2 boundary for any ECC or AES keys except as follows:

- If the operation that is being requested is a DSA or ECDSA Verify command.
- If the public key is an ephemeral key that is being used in a key establishment operation.
- If the public key is a static key that has been previously optimized through the use of a table or other optimization variables that were computed within the FIPS

140-2 implementation boundary for that key. The table may be cached outside the FIPS 140-2 implementation boundary, but it must be authenticated by the implementation prior to being used, e.g., using a keyed hash such as HMAC and a symmetric key that is kept exclusively within the cryptographic module. The symmetric key(s) used for that purpose shall be protected to the same extent and in the same manner as personal-use private keys.

Restrictions regarding key usage:

SP 800-56 restricts the use of a static key to only one operation, i.e., either encryption or signature. While we agree that this SHOULD be done in general, we believe that enforcing this requirement at all times would cause some significant problems for the industry. The logic is a bit convoluted, so let us cite chapter and verse:

1. Paragraph 5.6.4.2 (4) specifies that "one static public/private key pair **shall not** be used for different purposes (for example, a digital signature key pair **shall not** be used for different purposes (for example, a digital signature key shall not be used for key establishment or vice versa.)
2. Paragraph 5.6.3.2.2 specifies that "the recipient of a static public can obtain assurance from a trusted party (trusted by the recipient) that the owner of the static public key has provided the trusted party with evidence of the owner's possession of the static private key through explicit key confirmation (as described in Section 5.6.3.2.1 above), with the trusted party in the role of recipient.
3. Paragraph 5.6.3.2.1 in turn provides several key establishment schemes that can be used to carry out a key agreement, with a reference to Section 6 and 8 for details.
4. Section 8 deals with key confirmation, which can be either unilateral or bilateral, but in the case of Full MQV, (cofactor) One-Pass Unified Model, One-pass MQV, and (Cofactor) One-Pass Diffie-Hellman requires that the key confirmation recipient serve as the key agreement initiator.

The problem we perceive is that to the best of our knowledge, most CAs, and specifically both our PKI 6 product and the Microsoft Windows 2003 Server CA operating in conjunction with our Signal Identity Manager, process certificate requests through a queue mechanism. Someone may appear before a Registration Authority, confirm his identity, generate a key pair on the token, and then leave, with the certificate being generated in the non-real-time, for example after having run various credit checks, national agency checks, etc. Similarly, a user may request a certificate through a browser, but the certificate may not be issued immediately.

So non-real-time certificate issuance is a real requirement, but if proof of possession of the private key must be provided through an on-line key confirmation step that is initiated by the CA, or even the RA, then this may not always be practical, because the end-user may no longer be connected to the system.

The most common way around this problem is to have the user digitally sign the certificate request with the private key that corresponds to the public key, even if that key will later be restricted (in the certificate) to a particular key usage. Granted that this approach will not work for a pure Diffie-Hellman (FFC) solution, but it does work for RSA and for ECDSA keys.

Our recommendation is that the requirement to separate key establishment vs. signature key usage be softened to SHOULD**,** at least from the perspective of a hardware cryptographic provider. Instead, these requirements should be enforced at the protocol level, e.g., through the use of appropriate key usage bits in an X.509 certificate. That approach will permit a much finer-grained approach, allowing the separation of keys for nonrepudiation, "ordinary" signatures, authentication only keys, key exchange (encryption) keys, key archive keys, SSL keys, etc. If this approach is adopted, the problem will go away, because the certificate doesn't exist until it has been signed by the CA, and hence the policy enforcement issue would be moot.

At a minimum, we would suggest making an exception to this rule when providing proof-of-possession for ECC keys to a trusted third party. Or (worst case) we would suggest reverting back to the previous language that allowed proof–of-possession to be provided by distributing the certificate in encrypted form, requiring it to be decrypted and returned before it could be used. Note that this method is allowed under 5.6.3.1 (2).

Where is SP 800-56 to be enforced?

The previous issue raises a more general philosophical problem. Where is the enforcement of all of the various requirements SP 800-56 to be done? Consider a general-purpose operating system environment consisting of a plug-in hardware cryptographic token, a vendor-specific middleware or library, a higher-level protocol layer, and a consuming application. Where is SP 800-56 supposed to be enforced?

Consider the requirements of section 5.6.4.3 concerning ephemeral keys as an example. Must the destruction of the ephemeral key after it has been used once be initiated by the hardware cryptographic module, or can it be left to the application-specific protocol layer, e.g., an S/MIME protocol. After all, only the protocol layer and/or the application is likely to know whether there are going to be multiple DLC key transport operations taking place simultaneously, or within a short period of time. Should the cryptographic module be required to keep track of time, in order to enforce the "short period of time" requirement?

Similarly, section 5.8 says that the "shared secret **shall** be used in only one call to the KDF and shall be destroyed (zeroized) immediately following its use. The derived secret keying material **shall** be computed in its entirety before outputting any portion of it." But where must the shared secret be computed? Must it be within the same FIPS 140-2 cryptographic boundary as the private keys, or can it be computed elsewhere in the system? We believe the shared secrets should be confined to within the same cryptographic boundary as the private keys, whereas the *KeyingMaterial* referred to in

section 7 regarding DLC Based Key Transport may have to be exported from a hardware cryptographic module, e.g., a smart card, to a software module for efficient message or session-level encryption.

(For that matter, what kind of "teeth" does SP 800-56 have, or is it ultimately going to have? Is it going to be required for FIPS 140-2 or 140-3 compliance? Alternatively, are these recommendations that should be imposed at a system level, either for a NIAP/Common Criteria certification, or for a formal system accreditation?

Is SP 800-56 compliance going to be mandated for federal procurements? Or is it going to continue to be a "recommendation"?

Harmonization of SP 800-56 with NSA "Suite B" algorithm definitions:

Table 2 does not include and/or agree with the Suite B set of algorithms defined by NSA. In particular, parameter set ED specifies a minimum MAC key size and *MacLen* of 192 bits. We believe that the minimum MAC key size for Suite B should be 256 bits for ECC keys in the range of 384-511 bits. And the minimum size for *MacLen* should be twice the key size, or equal to the minimum hash length, or 384 bits. Note that there is no Approved keyed hash function that will output 192 bits in any case.

Excessive number of Key Derivation Functions:

The Key Derivation Function is an important component of a key establishment protocol. Unfortunately, the various standards in this area have failed to come up with any kind of a common consensus as to how a KDF should be implemented, or even what the important inputs are. SP 800-56 attempts to bow in the direction of all of the various standards, and ends up specifying four different KDFs and two optional variations with mutual agreement, for a total of six.

SPYRUS strongly believes that the KDF function must be carried out on the same cryptographic token that contains the private keys, but six KDF functions is simply too many to implement on such a token.

We would prefer that NIST specify <u>one</u> KDF that is sufficient for all of the various protocols, and then issue defect reports against the various standards to get them to conform to one common standard. Particularly in the case of ECC technology, there simply aren't enough implementations in the field to have to worry all that much about backwards compatibility.

At the present time, SPYRUS intends to implement a single KDF, the concatenation KDF without any variants, on our ECC tokens, as we feel that should be sufficient.

Ambiguities in the Key Derivation Functions:

The specification of the *ContextID* field in the KDF is too vague and meaningless to be very helpful to either implementers or standards writers. We feel that Appendix B should be revised to clearly identify the security threat that the contextID field is intended to address, and in particular the nature of the identifiers ID_U and ID_V.

In the absence of a compelling reason to do something different, SPYRUS recommends that D_U and ID_V be the hash H of party *U*'s encryption certificate and party *V*'s decryption certificate, respectively. If the Originator (party *U*) does not have or does not use an encryption certificate, e.g., in the case of One-Pass Diffie-Hellman used for S/MIME, then the Originator's decryption certificate may be used, since in nearly all cases the Originator will also be a recipient of their own e-mail message. If neither party has or uses a certificate, e.g., in the (Cofactor) Ephemeral Unified Model, then the user's name, handle, IP address, or other identifier shall be used in order to assure the Originator and the Recipient that they are communicating with the desired party.

We would prefer that the case of two fixed-length bit strings be dropped, and replaced with the single, more general case of two variable length strings, even if the lengths are equal in a particular case. The calling programs will have to pass the length in either case, but in addition would have to specify that the variable lengths are in fact fixed.

In addition, the length and bit-order of the $IDlen_U$ and $IDLen_V$ must be specified in the document, since they are input to the hash function. At present, we are assuming they are 32-bit non-negative big-endian.

At present, HMAC seems to be the only approved hash function that will generate the required number of output bits. If so, it probably ought to be specified explicitly.

Definitions of Key Establishment OIDs:

We would like to see NIST define specific OIDs for the fourteen key establishment schemes. Those OIDs should be independent of the key length and curve types, which can be determined independently.

Likewise, we would like to see NIST define specific OIDs defined for the Key Derivation Function (or functions, if necessary). The OID must specifically define the hash function to be used, e.g., concatenationKDF-with-sha256-HMAC. If any variants are to be supported (and we hope they won't be), then a specific OID must be defined for each variant.

Reference Implementations:

There is an immediate need for reference implementations and test vectors. Vendors are already implementing these functions, and reference implementations should have been available months ago.

Order of derived keys:

Paragraph 8.2 (3) states that the provider parses *DerivedKeyingMaterial* into two keys, *MacKey* and *KeyData*, with *MacKey* || *KeyData* = *DerivedKeyingMaterial*. As written, that suggests that the first *N* bits of *DerivedKeyingMaterial* will be used as the MacKey, if and only if the optional key confirmation step is to be done. Otherwise, the first N bits will be used for the KeyData. Having an optional key come first in a string is a particularly ugly construct, particularly since it may not be known at the time of the key establishment and KDF whether the key confirmation is to be done or not. PLEASE reverse the order of these two fields, unless there is some really good cryptographic reason for not doing so.

ECC public key validation:

It may be worth a footnote to point out that step 1, verifying that Q is not the point at infinity, can be done by inspection if the point is entered in the standard affine representation.

In step 2, we would recommend testing that x_Q and y_Q are in the range [1, p-1] rather than [0, p-1], as there are potential attacks if the x or y coordinates are zero. Since the chance of that happening by accident is almost infinitesimal, such a key ought to be viewed with suspicion.

Typographical Corrections:

Paragraph 5.7.1.2, process 3, says $Z = x$. It should say $Z = x_P$ to agree with the rest of the line.

Finally, from a reader's point of view, it would be nice if the document could be split into two comparable documents – one dealing only with ECC, and the other dealing only with FFC. Since there would seem to be very little interest in any new FFC implementations, the utility of the FFC portion of the specification seems questionable.

Thank you for your consideration.

Bob

Robert R. Jueneman
Chief Scientist
SPYRUS, Inc.
www.spyrus.com
2355 Oakland Rd., Suite 1
San Jose, CA 95131
408-545-0107

Date: Thu, 01 Sep 2005 15:34:43 -0400
From: Bruno Couillard bruno@bc5tech.com

I know the following comments regarding the subject document are late, but I thought I should pass them on none the less. I've picked them up while reading the document, and they are not major. Here they are:

1. Page 43, section 5.7.1.2, under Process, bullet 3, in the equation Z = x, x should be x_p.
2. Page 45, section 5.7.2.1.2, second paragraph, last sentence. I believe (y_A) should be (y_B).
3. Page 46, section 5.7.2.3, under Process, bullet 2. There is an extra parentheses at the end of the equation.
4. Page 46, section 5.7.2.3.1, first sentence, invoking the FFC MQV should be invoking the **ECC** MQV
5. Page 99, figure 10, the subscript used for MacTag on the bottom arrow should be V and not U.
6. Page 105, figure 16. an extra arrow is missing below the second arrow and before the separation dotted line with the tag: $Nonce_U$ and pointing from U to V.
7. Pages 106 & 107, figures 17 and 18. In all other figure representing these techniques, the steps taking place during key establishment are separated from those taking place during key confirmation using a separation dotted line in each box. Since the $Nonce_U$ passing takes place during the key establishment phase, that arrow should be moved above the separation dotted line in both figures.

That is it for me. I hope these were helpful.

Cheers,

Bruno Couillard, P. Eng.
President & CTO
BC5 Technologies, Inc.
W: +1.819.243.0340
C: +1.819.743.0381