Public Comments on NIST Draft Special Publication 800-56A Revision 2

NIST received the following public comments on the draft Special Publication 800-56A "*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, Revision 2 (August 2012)*".

Most comments were received in Microsoft Word format. They are merged in one document. The e-mail addresses and telephone numbers are removed. One commenter submitted a PDF version file and requested to publish in its original version.

Cisco Systems

Legend (type of comment)

E = Editorial
G = General
T = Technical

	SECTION, SUBSECT & PARA.	TYPE	COMMENT	RESOLUTION
1	5.6.1.1 and 5.6.1.2	T	The sections specify both that the private key to be in the range [2, q-2] ***AND*** that it must be generated using an approved method found in FIPS 186, appendix B. However, both approved methods in FIPS 186 generate values in the range [1, q-1]. Should the methods in FIPS 186 be modified to generate values in the range [2, q-2], or are values in the range [1, q-1] acceptable?	1. Modify FIPS 186 to generate values in the range [2, q-2] Or 2. Allow values in the range [1, q-1] as acceptable
2	5.6.1.1 and 5.6.1.2	T	Sections 5.6.1.1 and 5.6.1.2, attempt to exclude private keys of the values 1 and q-1. What is the reasoning behind this? If the reasoning is that someone listening to the exchange might see the public values g or g^-1 and immediately be able to deduce the private key, it should be noted that an attacker could precompute the public keys corresponding to the private keys 123456 and q-123456; if he then listens to the exchange and sees those public values, he will also be able to deduce the private key. If this is the reasoning behind the exclusion, why exclude the values 1 and q-1, but not the values 123456 and q-123456, when the latter have precisely the same weakness?	

Daniel Brown

Certicom, a subsidiary of Research In Motion

Provisos

Overall, I would say that the 56A authors have done well in writing and revising 56A. Key establishment, and discrete logarithm cryptography, are difficult and complicated subjects. The 56A document strives to tackle most of the complex details of these subjects.

Unfortunately, I did not provide myself enough time to review 56A thoroughly, for which, I apologize. Any of my comments that are incorrect, which may be likely, should be disregarded.

Text in quotes is taken from 56A. The red text below is suggested for insertion (or substitution) into 56A. I have opted not to quote this replacement text, because although it is an alternate text for to 56A to say, (i) it is not said in 56A, and (ii) the red styling is probably sufficient to delimit the replacement of substitution text, but if not they I can clarify). I am not sure how much more convenient for the 56A this red text convention is, but it helps remind me to make more constructive comments.

Technical Comments

General Technical Comments

None.

Specific Technical Comments

1. In the Static Unified Model scheme (as in Section 6.3.2, Page 108), if $Nonce_U$ is merely non-repeating but predictable, such as a sequence number, then the following harmful outcome could happen, whereas it would not happen if $Nonce_U$ were actually random. Suppose that in the future, an adversary Eve may somehow obtain: entity U's static key; and a ciphertext sent between U and V that were encrypted using the DerivedKeyingMaterial; but no record of the $Nonce_U$ that U to sent to V (say, because both parties destroyed the nonce and Eve was not present at the time of the key establishment). The harmful outcome is that Eve can decipher the ciphertext using U's static private and exhaustive searching the relatively small sequence number space, such as the time values, for the value of $Nonce_U$. In this setting, if $Nonce_U$ had been fully random, then Eve would not have been able to decipher the ciphertext. Accordingly, 56A should clarify this distinction to the user of 56A rather than leave the reader to determine what kind of nonce is needed here. I suggest putting a requirement that "$Nonce_U$ shall be a random nonce".

Editorial comments

The following comments are merely editorial. Their intent is, at most, to help clarify 56A, and at least, to help beautify 56A.

General Editorial Comments

The following editorial comments generally affect either the whole document, or refer to the addition of more material (so, either do not any specific part, or else refer to very many specific parts).

1. It would be nice if there were hyperlinks to navigate through 56A's cross-references. As a substitute, I've been using the PDF bookmark features (see also below).
2. It would be nice if there were a section, perhaps even a table, on compatibility of 56A with other major standards which use key-establishment, such as TLS, PKIX, CMS, WiFi, Bluetooth, and IKEv2. I believe that NIST has special publications devoted to some of these, in which case 56A could point to these special publications. Since I have not done a thorough review yet, I am not sure which of these common protocol standards are compatible with 56A. To elaborate, it would be nice if a table could translate the TLS ciphersuites into 56A's C(*,*) system, because then an implementer could, assuming compatibility of 56A and TLS, more easily ensure that a TLS implementation meets the 56A requirements and recommendations.

Specific Editorial Comments

The following comments refer to some specific parts of 56A. Some of the comments may generalize to other similar parts of 56A.

1. Page 33 says "Note: ANS X9.62, rather than ANS X9.63, specifies the most current ...", which could be incorrectly misinterpreted to mean that ANS X9.63 is out-of-date. Currently, ANS X9.63 has been updated and refers to ANS X9.62 for methods of generating ECC domain parameters. So maybe the note could be clarified to Note: ANS X9.63 currently refers to the ANS X9.62 for methods of ECC domain parameter generation.
2. Page 33, Table 2, uses some self- notations such as 160-223 and 512+. Although this notation is certainly self-evident for the given context, and perhaps even customary in non-technical contexts such as Table 2, it in inconsistent more the traditional technical mathematical notation (with – being minus, and + being plus) used elsewhere in 56A. It also misses an opportunity to the use the interval range notation [a,b] defined in 3.2. So, I would suggest, replacing 160-223 with [160, 223] and 512+ with [512, ∞] (with the understanding that ∞ is not allowed under the definition in 3.2 because it is not an integer, not to mention nonsensical and impractical). If this change is made, it might further help to adjust the wording of the row header to Interval of bit lengths of the ECC subgroup order n.
3. The symbol Ø for the point at infinity on page 51 looks slightly different (more bold) from the similar symbol Ø on Page 33 and elsewhere. Perhaps un-emboldening its format would resolve this discrepancy, as in Ø.

4. Page 98 has an "Error! Reference Source not found."
5. Page 108, Action 2, says "parameters D, U's static private key ds,U, and V's static". Some quibbles with this phrasing (and similar elsewhere in 56A) are the following.
   a. The part "parameters D, U's static" is hard to parse on a first read: is U part of the parameters? I suggest: parameters D, entity U's static, where entity has been inserted.
   b. Variable U appears twice in this phrase but only once in italics. I ought it to be same style throughout. It should be italics, because it is a variable, not a word, abbreviation or unit, so: parameters D, entity U's static, where U has changed to italics U.
   c. The word "and" is in italics. Is this just a formatting typo? If not, is it for special emphasis? If so, is there some more specific action intended to be taken in regards to this emphasis?
   d. The second comma in "$d_{s,U}$, and" seems to be a subscript (and italics), though it ought to be a regular comma, as it is separating items in a list at the level of regular text. So, I suggest: key $d_{s,U}$, and V's, where the second comma is removed from the subscript style, italics are removed from the word "and", giving and, and the variable V is in italics *V*.
6. The PDF bookmarks for the appendices seem a little wrong (A, B, and D are missing, with C included as part of Section 10.)
7. Page 33 top paragraph has "GF(q)" and "GF(q)", which is inconsistent in the style of parentheses. I recommend not using italics for parentheses (even in formulae): as in *GF(q)*.
8. Page 33 top paragraph uses quotes ""+"" and ""point at infinity"", and so does Section 5.4, with ""negligible"" and ""freshness"". (Note that because I've tried to quote all my quotes of 56A, there are double comments here; so, the outer quotes are mine, and the inner 56A's.) Do the 56A authors disagree with these terms, and thus quoting them, without acknowledging who uses these terms? Or, are the quotes merely for emphasis, or to introduce a new definition for a term. Elsewhere, italics seems to be used for such emphasis of introducing a new term, as in "h is called a cofactor" again on Page 33. Notice the italics for emphasis can sometimes be confused with the italics for multi-letter variables. Given that 56A has a list of definitions in Section 3.1, I recommend adding all the quoted or italics terms (or symbols) to the appropriate list, and then reverting the terms in the main body back to the normal font style of the surrounding text, without any quotes. So, the examples above would become +, point at infinity, negligible, and freshness, respectively. (Ideally, with a cross-reference linking back to the definition!)
9. Page 23 again uses quotes again in ""Vegetable.gardner123"", but here the 56A quoting is much closer to actual quoting, because it is a reference to a string outside the spec. It is also much akin the string data type literal quotation syntax of many programming languages. This quoted term should certainly be treated differently those discussed above: it should not be added to the list of definitions. As I noted, the quotes here are more acceptable, but it would be preferable to use

another common convention such as a monospaced font, as in `Vegetable.gardener123`.

10. Page 32 again uses quotes in "shall be "1"", but soon later uses an unquoted one with the same meaning in "bit 1 shall be 1". This is inconsistent. I suggest removing the quotes on the first instance and changing it to shall be 1.
11. Throughout the document, curly braces are used in three ways: (i) to delimit sets, as in Page 31, "{*0*, *1*, …, *p-1*}"; and (ii) to indicated optional elements of a list, as in Page 35, "*p*, *q*, *g*{, *SEED*, *pgenCounter*}, *G*, *n*, *h*", and (iii) as in Page 60, "*PartyVInfo* {|| *SuppPubInfo*}". For completeness, these notations should be added Section 3.2. The notations (i) and (ii) do not require much action from the implementation, but (iii) does. An entity must agree on the concatenated string with another entity, and if the optionality of these portions of the string is negotiated in real time, it must occur outside the simple conveyance of the bit string._

Thales e-Security

Legend (type of comment)

E = Editorial
G = General
T = Technical

ID	SECTION, SUBSECT & PARA.	TYPE	COMMENT	RESOLUTION
1	5.6.2.2 Para. 2, Bullet 1	E	The specific requirement for a public key recipient to assure the validity of a public key excludes the use of any caching mechanisms such that multiple exchanges with the same public key owner could be optimized by merely determining if the presented public key has already been validated using the referenced methods for key verification.	Add an option to allow the previous validation of a public key to be acceptable for meeting the requirement. An alternative option would be to extend the TTP paradigm to include an implicit 1st person trust, whereby a previous validation would be treated as 'trusted' by the 1st party (rather than a third-party).

2	5.6.2.2.1 Para. 1, Bullet 1	E	Similar to the comment above, this assertion excludes the situation where we have previously validated this static public key.	Extend the referenced sections to include the ability to reuse the result of a previous validation.
3	5.6.2.2.2 Para. 1, Bullet 1	E	Similar to the comment above, this assertion excludes the situation where we have previously validated this static public key.	Extend the referenced sections to include the ability to reuse the result of a previous validation.
4	5.7.1.1 Process Section	E	It is unclear the value added by including the requirement to convert z to Z using an integer-to-byte-string conversion routine. This seems too prescriptive.	Remove conversion to a byte string, including the use of Z.
5	5.7.1.2 Process Section	E	It is unclear the value added by including the requirement to convert z to Z using an integer-to-byte-string conversion routine. This seems too prescriptive.	Remove conversion to a byte string, including the use of Z.
6	5.7.1.2 Process #2	E	"…computation of Z (including z)" comment does not make sense in this context as 'Z' and 'z' have not yet been defined.	Should be rephrased as, "…used in the attempted computation of P and output an error indicator."
7	5.7.1.2 Process #4	E	Inconsistent lists of intermediate variables; P has been used as an intermediated but not listed as one to clear.	Either consistently enumerate intermediates to be cleared, or leave as a generic statement about clearing all intermediates.
8	5.7.2.1 Process #7	E	It is unclear the value added by including the requirement to convert z to Z using an integer-to-byte-string conversion routine. This seems too prescriptive.	Remove conversion to a byte string, including the use of Z.

9	5.7.2.1 Process #6 & #8	E	Inconsistent lists of intermediate variables; Z has yet to be calculated, and it is missing explicit intermediates other than z.	Either consistently enumerate intermediates to be cleared, or leave as a generic statement about clearing all intermediates.
10	5.7.2.3 Process #4	E	It is unclear the value added by including the requirement to convert z to Z using an integer-to-byte-string conversion routine. This seems too prescriptive.	Remove conversion to a byte string, including the use of Z.
11	5.7.2.1 Process #3 & #5	E	Inconsistent lists of intermediate variables; Z has yet to be calculated, and it is missing explicit intermediates other than z.	Either consistently enumerate intermediates to be cleared, or leave as a generic statement about clearing all intermediates.

Dr. René Struik

Struik Security Consultancy

Reference: NIST SP 800-56A – Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, Draft Revision, August 21, 2012

Review notes:

1. Summary

2. Review comments

2.1 General comments

2.2 Technical comments

- §5.6.1.2, p. 35: While the specification does specify the formula according to which ordinary discrete log key pairs are to be generated (§5.6.1.1, p. 35, l. 6), the corresponding group laws for elliptic curve arithmetic (see, e.g., §3.1.2 of [7]) seem to be missing.
- §6.1.1.4, p. 77: Note 2 seems to refer to notions that apply to the MQV scheme with ordinary discrete log groups, rather than elliptic curve groups (presumably, an editorial glitch).
- §C.1, p. 129: The integer-to-byte string conversion routine does not specify the bit-to-octet ordering.
- §C.2 p. 129: The field element to byte-string conversion

- §C.3, p. 129: The field element to integer conversion should make explicit the (currently implicit) assumption that polynomials (α is a binary polynomial, after all) are represented with highest-degree term "on the left", i.e., $\alpha = s_1 z^{m-1} \ldots + s_{m-1} z + s_m$.
- §C, p. 129: Compared to conversion routines specified in, e.g., Section 2.3 of SEC1 [5] (which are cross-referenced in many cryptography-related IETF RFCs), several conversion routines seem missing, including (a) integer to field element; (b) bit-string to byte-string; (c) byte-string to bit-string; (d) elliptic curve point to byte-string; (e) byte-string to elliptic curve point; (f) byte-string to field element; (g) byte-string to integer. While not all of these may be strictly required, most are, if only to unambiguously define the assurances in §5.6.2 (required assurances) or to unambiguously specify strings included with key derivation (§5.8) and with key confirmation of key agreement schemes (§5.9) and of key transport schemes (§7.2). We give some examples (while not being exhaustive):
  - §5.5.2, p. 34 (domain parameter validation: for prime curves, this relies on integer computations (see ANSI X9.62, §5.1.1.2) and would benefit from conversion (b) above. For ordinary discrete log groups, a similar remark applies.
  - §5.6.2.3.2, p. 46-47 (public key validation): this procedure takes as input a candidate ECC public key $Q=(x_Q,y_Q)$, which, witnessing the processing steps, would be an octet string, with implicit assumptions on how to convert this into an elliptic curve point, but the corresponding conversion (e) is missing. A similar remark applies to §5.6.2.3.3, pp. 47-48.
  - §5.6.2.3.2, p. 47: the input to this procedure presumably is a candidate ECC point $Q=(x_Q,y_Q)$, in affine representation, but the first step of the procedure checks whether this is the point at infinity, which does not have this format. This suggests that the input to the procedure should be a byte string that would allow for representing any ECC point.
  - §7.2, p. 115 (key confirmation with key transport scheme): the *MacData* field depends on $EphemData_V$ *and* $EphemData_U$, which may be both ephemeral public keys, but the representation hereof is not specified (thus, requiring inclusion of conversions (b), (d), and (e) above, as other specifications (e.g., ANSI X9.63, SEC1, IETF RFC) do. This typically includes specifying the octet representation of affine points, point-at-infinity, compressed points.

  While the intention could have been to leave the specification of missing conversion routines to implementers, it is more likely that this is an omission in the specification, since not mentioned as prerequisite/assumption underlying the use of key agreement schemes (§6.1.1, pp. 70-71; §6.1.2, p. 81; §6.2.1, p. 85-86; §6.2.2, pp. 98-99; §6.3, pp. 105-106). Leaving out conversion routines would be contrary to the goal of fostering interoperability, may increase implementation cost, may lower implementation security posture (e.g., if these routines are implemented outside a trust boundary and either routines or data these act upon can be manipulated), and seems out-of-sync with granularity level of other NIST specifications (where e.g., bit-to-byte ordering is specified). When specifying point compression, the specification should not require conversion hereof to other formats, when this is not strictly required from a computational perspective (e.g., with ECDH, scalar multiplication can be implemented with x-coordinate only arithmetic).
- §A, pp. 126-127: the references should be split into normative and informative references (where those related to specifications [ANSI, FIPS, NIST] are normative).

2.3 Editorial comments

- §6.3.3.3, p. 112, l. 2: fix the broken cross-reference, here presumably to Fig. 18 on the same page. (Some other cross-references also need fixing, e.g., §6.3.3.3, p. 113, l. 2 [presumably to same figure] and §6.2.1.5.3, p. 98, l. 2 [presumably to Fig. 12 on p. 97].)
- §A, p. 126: NIST SP 800-56C was published November 2011.

- §A, p. 126: NIST SP 800-57 has several parts (Part I, Part II, Part III), with several updates/revisions since 2005. It is not clear which version and which part the reference refers to.
- §C.1, p. 129, l. 3-4: I would suggest making the remark on use of this conversion routine with the FFC scheme a separate note (after all, this routine is also used for conversion of prime field elements to octet strings (cf. §C.2 on the same page)).
- §C.1, p. 129, second item: The formula is somewhat awkward, since the summation is taken over the integers i=1, …, n (and certainly does not hold for each integer in that range). If one wishes to display formulas inline, why not just putting $C = 2^{8(n-1)} S_1 +\dots 2^1 S_{n-1}+ 2^0 S_n$. A similar remark applies to, e.g., §C.3, third item.
- §C.2, p. 129, first item: Replace “of length n bytes” by “of length n”: the length of an octet strings does not have units. A similar remark applies elsewhere, e.g., in §C.2, second item, where one should replace “of length m bits by “of length m”.
- §C.2, p. 129: the conversion routine for binary extension fields could be described in a simpler way, as follows: (a) left-pad the bit string $s_1 s_2 \dots s_m$ with $8n$-m zero bits; (b) partition the resulting bit string of length $8n$ as the right-concatenation of n substrings $S_1 \| \dots \| S_n$, each of the same length.
- §D, p. 133: The rationale for removing the old Appendix A includes the statement that ANSI X9.63 has been updated to be consistent with the current draft specification. Since the specification references ANSI X9.63-2011 in §A, this seems to assume that no changes to the current draft (August 2012) would be made that could possibly impact alignment with an external specification published last year (2011). This seems to tie NIST hands behind the back…

References

[1] B.B. Brumley, N. Tuveri, “Remote Timing Attacks Are Still Practical,” IACR ePrint 2011-232.
[2] J. Fan, B. Gierlichs, F. Vercauteren, “To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order,” CHES 2011.
[3] L. Goubin, “A Refined Power Analysis Attack on Elliptic-Curve Cryptosystems,” PKC 2003.
[4] T. Izu, T. Takagi, “Exceptional Procedure Attack on Elliptic Curve Cryptosystems,” PKC 2003.
[5] NIST SP 800-56A, “Recommendation for Pair-wise Key Establishment Schemes Using Discrete Logarithm Cryptography”, Revised, March 8, 2007.
[6] SECG, *SEC1: Elliptic Curve Cryptography*, Standards for Efficient Cryptography, Version 2.0, May 21, 2009.
[7] D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2003.

Hugo Krawczyk

The comments are submitted in a PDF file starting from the next page.

Comments on NIST's August 2012 Revision of SP 800-56A "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography"

Hugo Krawczyk*

November 8, 2012

In 2005, I reviewed the version of this document at the time and provided very detailed comments on many of its aspects ranging from presentational issues to core cryptographic considerations. The report, included here for reference, pointed out to serious weaknesses in the specified cryptographic schemes. In particular, it criticized the preservation of weak cryptographic schemes in the name of compatibility with old documents (e.g., ANSI X9.42) while ignoring the significant advances in the area of key exchange research. It was suggested that similarly to the case of hash functions, weak key exchange schemes should be gradually phased out in favor of stronger schemes.

Unfortunately, 7 years later, virtually all these recommendations have been ignored and the same weak mechanisms are being standardized in 2012. This is particularly regrettable in an area where large deployment is just starting, due to the increasing attractiveness of elliptic curves, and there are not too many "legacy systems" to be tied to. It is also an area where cryptography is quite well understood with very active research and scrutiny of schemes. One can only wonder why NIST, who has shown commendable openness to modern schemes when adopting hash functions, block ciphers, modes of operations, etc., would insist in standardizing sub-optimal (and even broken) key exchange schemes. This helps perpetuating these schemes, especially given the mandatory aspect of these standards when it comes to US government validation and the fact that industry looks at NIST's cryptographic standards as a model to follow.

I hope it is not too late to address the many shortcoming of this document. The attached report seems as relevant as it was when written in 2005 and I ask NIST to give it serious consideration.

Here is a listing of some of the main issues still remaining in the current revision (to which I refer as 56A).

- Protocol UM (in its FFC and ECC variants) is broken. Blake-Wilson, Johnson and Menezes showed in 1997 (that's 15 years ago) that the protocol is vulnerable to trivial interleaving and known-key attacks. The attacks are relevant in any setting where a pair of parties may have more than one simultaneous key exchange sessions (as it would be the case for simultaneous SSL connections or IPsec tunnels), and are not prevented by any of the auxiliary mechanisms contemplated in 56A such as proofs of possession or including identities under the KDF.

  The interleaving attack is solved if the ephemeral DH values are included under the KDF (and if the latter is modeled as a random oracle). To the very minimum, the UM protocols in 56A should be upgraded to provide that defense.

*IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA.

- Protocol MQV has some attractive features but has not been proven secure. More significantly its security may depend on external mechanisms such as CA-performed proofs of possession that are unrealistic in many practical settings. There are known alternatives, such as HMQV, offering the same attractive features of MQV but with provable (and widely scrutinized) security, better performance, and without requiring external (often unrealistic) mechanisms such as proofs of possession (see below) or other assumptions on certification authorities.

- 56A mandates proofs of possession (PoP) for all its protocols. In a PoP a user claiming to own a public key needs to prove it also knows the corresponding private key. According to 56A, CA's should require such proofs at the time of certifying a public key. As explained in detail in the attached 2005 report, designing secure PoP mechanisms is very tricky. In particular, the soundness of the PoP mechanisms defined in 56A is questionable as they present the following "circular paradox". According the specification, to prove PoP, the CA will run one of the 56A KE schemes with the user. But what's the security of these PoP mechanisms given that the KE scheme used to prove it is in itself not secure without a prior PoP?

  To further exemplify the problems with the PoP requirement, consider the case where parties A and B run a KE scheme, but A cannot be assured that B's public key was tested for PoP. In this case, 56A (Section 5.6.2.2.3.2) mandates A to get assurance of B's possession of the private key by running a KE scheme with B that includes a key confirmation step. The idea is that if an attacker E wants to claim possession of B's public key without knowing the corresponding private key, E will not be able to complete the key confirmation step. However, in a model where an attacker may be able to obtain ephemeral session information but not the static private key of B, E can complete the key confirmation step without ever learning B's static private key. This allows E to claim ownership of the same public key as B by playing man-in-the-middle between A and B.[1]

- Besides being hard to do them right and even harder to guarantee that CAs will actually perform them, proofs of possession present other severe limitations such as the inability of a user to self-certify his DH public key. Specifically, consider the case of a user U who has a certified signature key (a much more common case than having certified DH keys). For engaging in KE schemes as those defined in 56A, U chooses a DH public key, p, and signs it with his own signature. Thus, U can certify p by showing the signature on p and the certificate on the signature public key. Clearly, such certification would violate the 56A specification. In summary, mandating CA-run PoP is unrealistic in many cases, it kills natural usage scenarios with self-delegation, and is likely to *hinder adoption by the industry at large.* Such resistance would be further justified by the mere fact that both UM and MQV are not secure even with PoP. As said, UM is broken by interleaving attacks even with PoP, and MQV is vulnerable to UKS attacks even with PoP's (Kaliski showed that in 1998). The better alternatives to MQV

[1]The attack: A sends to E the value g^x who forwards it to B as coming from A. B responds with g^y and $MAC_K(B, A, g^y, g^x)$ where K is derived from the ephemeral shared secret Z (computed by B on the basis of g^x, y, A's public key and B's private key). Now, E breaks into B's ephemeral session state to find the ephemeral shared secret Z and sends to A the value g^y and $MAC_K(A, E, g^y, g^x)$. According to 56A, this should convince A that E knows B's static private which is false in this case as E has never learned this key. Note that even including the identities under the KDF does not prevent the attack. In the latter case A and B would derive different session keys but E could then make this session abort (e.g., by never sending to B the last message from A) to prevent A from learning this mismatch. But the damage has been done: A believes she has run a successful PoP with E and will accept B's key as belonging to E in future KE runs.

mentioned before also solve the PoP issue: they provably dispense of the need for PoP's and other costly checks by the CA.

- The document lacks rationale for most of its design. One exception is Section 8 (new in the current version) where some rationale is presented to help a user choose between all schemes. Unfortunately, this is a very superficial and faulty account which leaves out some essential information. For example, Section 8.1 states a fundamental security property of C(2e,2s) schemes: that "each party has assurance that no unintended party (i.e., no parties other than the owners of the corresponding static key pairs) can compute the shared secret, Z, without the compromise of a static private key." The only way to understand this is with respect to an eavesdropping adversary. Active attackers, however, may not need to learn the parties' private keys. In the case of UM, it suffices to learn the shared key Z_s (defined as the DH operation on the parties' static PKs) which functions as a long-term shared symmetric key and may be cached for long periods and afforded less protection than a static private key. Particularly puzzling is the absence of *forward secrecy* as a main consideration when choosing a DH exchange (with ephemeral keys).

- The other case where more detailed rationale is given is in Appendix B, "Rationale for Including Identifiers in the KDF Input". This is the only place where works from the cryptographic literature are cited. This is strange as the need to bind identities to an exchanged key is a well-known pillar of KE security and using the KDF as a way of doing this is quite standard. Actually, for both UM and MQV it would be better to mandate the inclusion of identities under KDF. At least in the case that the KDF is modeled as a random oracle this would help against UKS attacks.[2]

- A few more comments:
  - See the criticism of the naming of schemes in attached 2005 report (another puzzling legacy from old standards) which remains as valid as before.
  - The current version of 56A still carries the somewhat enigmatic requirement (Section 5.8) that "Non-secret keying material (such as a non-secret initialization vector) shall not be generated from input that includes the shared secret." If revealing part of the output of a KDF endangers the non-revealed parts then that KDF is plain insecure.
  - Page 57: max_H_inputlen: what's the idea? How should it be used?
  - In the discussion of nonces in Section 5.4, the birthday paradox needs to be discussed and random nonces made long enough to prevent collisions.
  - It would be a good idea to add a MAC from sender to receiver in C(1e,2s) one-pass protocols. It achieves full forward secrecy for the sender (against both active and passive attackers) without any computational burden. (This technique and its security is shown in the PKC'2011 paper "One-Pass HMQV and Asymmetric Key-Wrapping" by Halevi and Krawczyk.)

[2] I have criticized in the past a KDF scheme proposed by NIST where identities were mandated for ALL key exchange protocols. I pointed out that this was too broad since some KE schemes do not need the identities under the KDF and some cannot possibly include them since the identities may not be known at the time of KDF application (e.g., IKE). But when describing specific schemes as in this document I do not see a reason not to include the identities under the KDF as part of the scheme's definition.

- Finally, the attached 2005 report makes some points about the importance of provable security that are worth re-reading. Sometimes, weaknesses spotted via technical analysis are disregard as theoretical-only. In this sense it is instructive to consider the history of TLS and all the attacks against the protocol that started as "theoretical-only" and slowly but surely found their way into real-world attacks requiring changes to the protocol (or in the way it is used). Similarly, it is easy to disregard UKS attacks or the inappropriate implementation of PoP techniques as theoretical. They may stay as such only for as long as these protocols are not used in practice. If these protocols ever become ubiquitous, these weaknesses will have practical consequences. For the benefit of preventing these problems and to aid adoption of DLC-based KE protocols, the schemes in 56A, and their description, need to improve significantly. It is not too hard to do it, just start by getting rid of obscure legacy ties to old specifications.

A 2005 Report on SP 800-56

Attached.

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

[*]IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA. Email: hugo@ee.technion.ac.il

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 plaussible 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 re-uses q to denote the order of the underlying ECC field). We refer to elements $x \in Z_q$ as DH exponents and to group elements of the form g^x 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: Z_s and Z_e. The first is obtained as the DH computation applied to the pair of static public keys (i.e. $Z_s = g^{ab}$) and the second as the DH computation applied to the pair of ephemeral DH values (i.e. $Z_e = g^{xy}$). 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).

[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.

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 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 $d = H(g^x, ID_b)$ and $e = H(g^y, ID_a)$ where ID_a and ID_b 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) $x^q = 1$ or that (ii) $X^h \neq 1$ 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 ($|h| \leq 32$). 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 $X = g^x$? 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 Z_s 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 $C(\cdot,\cdot)$ 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., Håstad, 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-ietf-ipsec-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 Log-based 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 public-key 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.