Evaluation of Security Level of Cryptography

ECMQVS (from SEC 1)

Phillip Rogaway **Mihir Bellare** **Dan Boneh**

January 16, 2001

Contents

This document analyzes the ECMQVS from SEC 1. The analysis is based on the SEC 1 documentation (version 1; 20 September 2000). Additional guidance was provided by the paper of Law, Menezes, Qu, Solinas, and Vanstone [LMQSV].

1 Summary

The ECMQVS (Elliptic Curve MQV Scheme) of SEC 1 is a widely standardized key-agreement scheme based on elliptic-curve cryptography. Our high-level findings are as follows.

1. **Over-generalized trust model**. The ECDHS attempts to support a variety of trust models, not fixing whether the two keys that each party uses are static or ephemeral. Of the $2^4 = 16$ trust models, we believe that only one should be encouraged. Fixing the trust model would not only eliminate unlikely and often-unsafe possibilities, but it would allow considerably less vagueness in the exposition of the goals.

2. **Efficient.** ECMQVS is probably the most efficient scheme that has been suggested for the set of goals it was designed to meet. The cost is about 2.5 elliptic-curve multiplications for each party.

3. **Intrinsic limitations.** ECMQVS is a two-flow protocol in which the second flow is independent of the first. As such, it provides no key-confirmation; a party has no reason to believe that its intended peer has received the distributed session key. Additionally, two-flow protocols achieve a weaker form of forward secrecy. It may be desirable to support, as an option, architected methods for key-confirmation (ie., an optional component for the second flow and an optional third flow).

4. **Provable-security status.** It is suggested in [LMQSV] that provable-security results should be possible for ECMQVS. We doubt this. Two changes (indicated below) might help, but, even then, we have not worked out a proof, and doing so would not be a small matter. Further changes could well be needed.

   The assumption for a security proof would probably not be the Diffie-Hellman assumptions. We describe a new assumption, the "Linear Diffie-Hellman assumption" (LDH) that would seem to be necessary. We do not know if such an assumption is sufficient. We do at least demonstrate random self-reducibility for LDH, which perhaps improves the assumption's plausibility.

5. **Recommended adjustments.** It might be desirable to modify ECMQVS in a couple of ways: (1) We suggest that the function $\overline{\ }$ (which maps group elements to points of about $n/2$ bits) be replaced by a hash function. In

the absence of this change, the security of ECMQVS would seem to depend on a (conjectured) non-interaction between the specifics of the function $\overline{}$ and the structure of the elliptic-curve group. (2) We suggest that the inclusion of identities be mandated for the string *SharedInfo*. In the absence of this one gets an overly weakened notion of security or else one must assume the use of counter-measures which reach beyond the scope of the standard.

6. **The future.** We suggest that the technology is nearly in place that emerging standards for AKE could be both efficient and provably-secure (under strong definitions and weak assumptions). The time is due for this to happen.

Though ECMQVS comes with significant issues, the reviewers were more favorably inclined towards it than towards any of the other CRYPTREC key-sharing proposal.

2 The Scheme

ECMQVS is a reasonably simple scheme. Two parties, whom we denote[1] **A** and **B**, share valid EC domain parameters, including the base point G which generates a subgroup $\langle G\rangle$ of the elliptic-curve group E (over field F_q), this subgroup having prime order $n = \frac{|E|}{h}$, where $h \in \{2, 3, 4\}$.

Entity **A** has a (secret key, public key) of (a, A) and another (secret key, public key) of (x, X). Here $A = aG$ and $X = xG$. Likewise, entity **B** has a key pair (b, B) and another key pair (y, Y). Here $B = bG$ and $Y = yG$. The parties exchange their public keys: **A** gets B, Y and **B** gets A, X. The keys A and B are exchanged in an authenticated way: the association of these keys to **A** and **B** is supposed to be known (eg., by a certificated) by **B** and **A**. Now the parties compute a shared key of

$$\begin{aligned} S &= h(x + a\overline{X})(y + b\overline{Y})G \\ &= h(x + a\overline{X})(Y + \overline{Y}B) \quad (\text{for } \mathbf{A}) \\ &= h(y + b\overline{Y})(X + \overline{X}A) \quad (\text{for } \mathbf{B}) \end{aligned}$$

Here $\overline{}$ is a particular hash function that maps from EC points to numbers (we will define it in a moment). Now: if $S = O$ the party *rejects*. Otherwise, the shared session key is $K = H(s, \textit{SharedInfo})$, where *SharedInfo* is an optional, unspecified string and s is a string that encodes the x-coordinate of S.

We now define the hash function. We fix an injective map `xec` from the field F_q to the set of integers $\{0, 1, \ldots, q-1\}$, which is used to represent field

[1] SEC 1 names the parties U and V. Besides looking to similar, as do their lower-case versions, we wish to regard one party (**A**) as the initiator as one party party (**B**) as the responder. This semantics is not associated to U and V, and we do not want to create confusion. Additionally, the subscripts associated to the notation in SEC 1 seems cumbersome, so we will adopt the simpler notation given here.

elements. We let $f = \lfloor \log_2(n) \rfloor + 1$ denote the security parameter of the scheme (e.g., $f = 160$ or $f = 224$). Then define:

Function $\overline{P}$ (The input $P = (x, y) \in E$ is a point in the elliptic curve group)
$\overline{x} \leftarrow \texttt{xec}(x)$
Return $(\overline{x} \bmod 2^{\lceil f/2 \rceil}) + 2^{\lceil f/2 \rceil}$

For future reference, we also define the operator `xts` to take as input a point $P = (x, y) \in E$ in the elliptic curve group and return a binary representation of $\texttt{xec}(x)$. This is the function implicitly used already in mapping S to s.

3 Static and Ephemeral Keys

The expected way to use ECMQVS is for (A, a) to be a static key-pair (that is, it will be used for multiple session-key exchanges) and (x, X) to be an ephemeral key-pair (that is, it will be used for a single session-key exchange). Similarly, (B, b) will be static and (Y, y) ephemeral. However, none of this is mandated, or even discussed, in the SEC 1 text.[2] In fact, the specification clearly allows any of the key pairs to be static or ephemeral, and says

> [Goals achieved by ECMQVS depend on] whether or not U and V both contribute ephemeral key pairs. [SEC1, p. 47]
>
> Selection of an appropriate application of the scheme will likely depend on ... whether U and V are both on-line. [SEC 1, p. 72]

We believe that anything but (A, a) static and (X, x) ephemeral, (B, b) static and (Y, y) ephemeral, is a questionable usage mode, as we will discuss.

4 Goals

The SEC 1 document does not give much help in understanding the protocol's goals. It states, for example

> [ECDHS and ECMQVS are] designed to meet a wide variety of security goals depending on how they are applied .... [Goals] include unilateral implicit key authentication, mutual implicit key authentication, known-key security, and forward security, in the presence of adversaries capable of launching both passive and active attacks. [SEC1, p. 45]

Unfortunately, the document never explains what security goals is intended to follow if the scheme is applied in some particular way. Nor does it carefully

[2] Entity **A** is supposed to get B from **B** in an authenticated manner, while **B** is supposed to get A from **A** in an authenticated manner. One might conclude that A and B should be static keys with associated certificates. Or one might have concluded that A and B should be signed by **A** and **B**. Or one might not conclude anything!

describe all of the terms used.[3] The scope of "how it might be applied" is not discussed at a level allowing a user to understand what security goal will follow from what way to use ECMQVS.

That said, the supporting document [LMQSV] does better in explaining what are the protocol's goals. But the goals there refer to "Protocol 1" in that document—which is not exactly in harmony with the apparent goals of the SEC 1's ECMQVS, insofar as ECMQVS adds trust-model possibilities which are not a part of Protocol 1.

From [LMQSV] one gets that the main goal is very weak; paraphrasing the second paragraph of p. 1, the protocol is intended to provide *implicit key authentication*, by which the authors mean that *if* entities **A** and **B** are *honest*, then **A** is assured that no one other than **B** can possibly learn the value of a particular secret key S. The authors go on [LMQSV, p. 10] to say:

> We emphasize that lack of the unknown key-share attribute does not contradict the fundamental goal of mutual implicit key authentication — by definition the provision of implicit key authentication is only concerned with the case where [**A** and **B**] engages in the protocol with an honest entity

We are not sure if restricting attention to honest parties corresponds (in the formal model of [BPR00]) to assuming that the adversary who can not make Send queries, or that it means, instead, that she can only make Execute queries. Either way, this is simply *too weak a goal*: restricting attention to passive adversaries is not a good way to understand the goal for any authenticated key exchange. The essence of AKE is defeating active adversaries (this is what distinguishes the goal called Session Key Exchange, SKE).

The authenticated key-agreement goal, as one might like to describe it (still quite informally), might say

> Consider an active adversary whose capabilities include, at least, sending arbitrary messages to instances of entities and obtaining session keys from these instances. When a party **U** accepts a session key K as having been shared with a partner **V**, then at most **U** and **V** know any information about this key K, assuming that the adversary has not obtained this session key by having asked for it.

As simple as this sounds, the above is a high standard. And it is not achieved by ECMQVS, as Kaliski has shown [Kal98], and as subsequently discussed.

Despite considerable research in this area, it is not entirely resolved what exactly are the goals that a protocol like ECMQVS *can* achieve, which are the important goals *to* achieve, and what goals are actually that a protocol like this *does* achieve under different scenarios. We understand the basic goal to be along the lines of those defined in [BPR00].

[3] Informal descriptions are given for implicit key authentication and forward secrecy, the former perhaps not in-line with [LMQSV], as no mention of honest parties or passive attacks is made.

Trying to have a clearer description of the intended goals is not just of "academic" interest. Is the [Kal98] "unknown key-share attack" a "real" attack? To us, yes. To others, maybe no.

Some further goals include:

1. *captured session-key security.* Loss of a session key should not compromise other session. (We ourselves consider this to be a "basic" (necessary) goal of a key-exchange protocol.

2. *unknown key-share security.* When a party believes it has a shared session key with some entity U, it could not, in fact, have shared session key with some *other* entity, U'. (We again consider this to be a basic goal.)

3. *forward secrecy.* If static private keys are revealed this does not compromise previously-distributed session keys. (This is a highly desirable property.)

4. *Reveal$ security.* If an instance should have its private coins revealed, this will not compromise other sessions.

5. *dual contribution.* Neither entity can force the key to some preselected value.

6. *dictionary-attack security.* Suppose that the static private key a of **A** is derived from a possibly poorly-chosen secret (a password), and that **B** keeps the associated public key A of **A** secret. We would like that the adversary is unable to gain information about **A**'s secret key more effectively than interactively trying the most likely secrets, in order. In particular, off-line computation with time complexity proportional to the guessing-complexity of the password space used by **A** should not be effective in recovering **A**'s password.

The first of these goals is viewed as necessary and is believed to be achieved by ECMQVS. The second goal is problematic, as we will discuss. Forward secrecy is an important goal and seems to be achieved by ECMQVS. Security in the face of coin-revelations is a nice property that ECMQVS again seems to achieve. The fifth goal from this list does not seem important to us, but again it seems to be achieved by ECMQVS. The fifth goal is a very nice property which ECMQVS appears to achieve, and which, to the best of our knowledge, has not been pointed out before (perhaps because it has been more common to consider dictionary attacks under a weaker trust model, where **A** does not have a public key for **B**).

5 Efficiency

ECMQVS gets good marks for efficiency. The computational work associated to ECMQVS is well-approximated by looking at the number of multiplications in the elliptic-curve group that each party must perform. For the

(static/ephemeral, static/ephemeral) case that we focus on, the communications cost (ignoring static keys and their certificates) is a single ephemeral key in each direction.

For the (static/ephemeral, static/ephemeral) case one needs **2.5** multiplications, one of which may be performed off-line, before ones peer is contacted. (This 2.5 figure assumes that one has already checked any certificate needed to check the binding of a peer to his specified (static) public key.) Some alternatives: signing a DH key exchange would use 2 multiplications, a signature generation, and a signature verification; and encrypting "half" of a MAC key (the approach of SKEME) would use 2 multiplications, an encryption, and a decryption.

Of course it is not only the number of multiplications that matter. Another important factor is the size of the group where we are working. Without reductions in place, and without knowing their concrete security, it is impossible to draw firm conclusions. Nonetheless, a security level of 80 bits, that is, $n = 160$, is probably adequate in practice.

The SEC 1 protocol is quite open-ended in the groups that it allows, so, if desired, acceleration methods for characteristic 2 and methods specific to particular curves can be employed.

Kaliski observes [Kal98, p. 8] that the "original" MQV scheme can be implemented with just **2** multiplications. But these differences (2 multiplications vs. 2.5 multiplications vs 3 multiplications) may be less important than provable security.

6 Some Attacks

Let us examine some attacks on ECMQVS.

6.1 Attacks in the static/static cases

The (static/static, static/static) case. Suppose that both of the parties, **A** and **B**, contribute only static keys: all of (a, A), (x, X), (b, B), (y, Y) are static key-pairs. Then the ECMQV scheme does not achieve the most basic goal of a session-key distribution protocol, as the key which is exchanged between **A** and **B** is a *fixed* key associated to $(\mathbf{A}, \mathbf{B})$. The protocol should not be called an authenticated key-agreement protocol.

The goal of mapping two static public-keys to an associated secret (known to the parties with the corresponding secret key) is a worthwhile one. But it has almost nothing to do with authenticated key-agreement. And if this is the goal, one does not need ECMQVS. Instead, each party should use just one (static) key pair, (A, a) and (B, b) and the shared key would be $K_{\mathbf{AB}} = H(\mathbf{A} \,.\, \mathbf{B} \,.\, S)$ where $S = habG$. Omitting **A** and **B** is an invitation for trouble.

After two static public-keys are mapped to an associated key $K_{\mathbf{AB}}$, that key should not be directly used for cryptographic purposes like encryption or message authentication. Instead, a session-associated key-variant should be

used, for example, by setting $K = H(K_{\mathbf{AB}} \ . \ R)$ where R is a random value (not to be kept secret) selected by one of the parties. Omitting such a session-associated key-variant can be dangerous. Suppose, for example, that $K_{\mathbf{AB}}$ is used for CTR-mode encryption. The result is disastrous, as we are essentially reusing the pseudorandom pad with every message.

All in all, we can conclude the following.

Summary: **Using ECMQVS with all-static keys isn't a session-key distribution protocol, though it does do something. For what it does, better choices are available.**

The static/static — static/ephemeral case. Suppose now that one of parties contributes two static key-pairs, while the other party contributes a static and an ephemeral key-pair. This situation is not as bad as before: the session key K will vary with each run, due to the contribution from the ephemeral key. But there are still concerns (implicit in the trust model, not the specific protocol). First, if the adversary **Z** learns even a single session key K, then **Z** can impersonate the static/ephemeral entity to the static/static one: just use the recorded flow from the static/ephemeral entity along with the associated session key.

The trust model implicit in this case may be useful. It is suggested in the document that one party might not be on-line. Following this lead, consider an application like the sending of email: entity **A** wants to place a message M in the "mailbox" of entity **B**. Assume entity **B** is not on-line, but **A** already knows (or can obtain) the public key B for **B**. Then **A** will have to prepare a message that does not depend on interaction from **B**. An ephemeral key from **B** can't be used.

In this model there is the exposure already mentioned, if a session key is lost. But that may be irrelevant to the application. Forward secrecy also is limited. In particular, there can be no forward secrecy in the face of loss of b: necessarily this allows recovery of the message M sent by **A**. On the other hand, there can be forward secrecy in the face of loss of a, and that would be provided by ECMQVS.

Still, the entire example is a bit misguided. Fundamentally, entity **A** was not trying to create a session with **B**, so thinking of this process as a key-agreement scheme never made a lot of sense. Beyond this, the ECMQVS is unnatural and complex for this situation. In particular, we are forced to imagine **B** having two public keys: maybe (B, b) two times.

By being non-specific about the trust model it is difficult to explain simply and generically what the protocol intends to do. Overall, it seems a loss.

Summary: **It seems best to mandate for ECMQVS that each party uses a static key-pair and an ephemeral key pair. If other trust models are important, they should be singled out a protocol crafted specific to that trust model.**

We henceforth assume the static/ephemeral — static/ephemeral trust model.

6.2 The unknown key-share attack of Kaliski

Kaliski [Kal98] suggests the following clever attack on ECMQVS. The attack assumes that the adversary **Z** can obtain a certificate for any public key of her choice.

Suppose that entity **A** uses static key-pair (A, a) and ephemeral key-pair (X, x). Entity **A** sends A and X to **B**, along with a certificate Cert**A**A binding A to **A**. The adversary **Z** captures this flow and goes to the CA so as to obtain a certificate Cert**Z**E for the new public key $E = (G/\overline{X + \overline{X}A - G})$. (The reciprocal here is in Z_n). The adversary knows the corresponding secret key $e = (1/\overline{X + \overline{X}A - G})$. Now the adversary replaces x by $X_* = X + \overline{X}A - G$ and she replaces the certificate Cert**A**A by Cert**Z**E. The flow going back from **B** to **A** is not manipulated. Now observe that **A** will have a PID (partner ID) of **B** and a session key K derived from the shared elliptic-curve point

$$S = h(a + a\overline{X})(y + b\overline{Y})G$$

while **B** will have a PID of **Z** and the same session key K, as it is computed by

$$\begin{aligned} S &= h(X_* + A_*\overline{X_*}A)(y + b\overline{Y}) \\ &= h((X + \overline{X}A - G + (\overline{X + \overline{X}A - G}/\overline{X + \overline{X}A - G}) \cdot G)(y + b\overline{Y}) \\ &= h(X + A\overline{X})(y + b\overline{Y})G \end{aligned}$$

Thus **A** believes (correctly) she is talking to **B** while entity **B** believes (incorrectly) that he is talking to **Z**.

Documents [SEC1] and [LMQSV] refer to this type of problem as an "unknown key-share attack." It is regarded their as an essentially "optional" characteristics of a key-exchange protocol. For us, this is a central goal of key exchange: when you accept with a particular partner ID, only that named entity may have the shared session key.

The preferred solution to this problem is probably to include the names of the entities (or their certificates) in the scope of the key-derivation function H. We conclude the following:

Summary: **ECMQV is susceptible to unknown key-share attack. To overcome this, include identities of protocol participants in *SharedInfo*.**

Note that this attack is not thwarted even by moving to a stronger certification-authority model, where entities must establish that they know the corresponding decryption key of a public key that is being certified. Here **Z** *does* know the corresponding secret key.

An alternative countermeasure is to insist on key-confirmation.

6.3 Attacks intrinsic to 2-flow AKEs

This section discusses limits of authenticated key-exchange protocols under the assumption that there are only two flows, and the second flow is independent of the first. All versions of ECDHS fall into this category.

1. **No forward secrecy in the strong-corruption model.** In modeling the possibility of a party being corrupted there are a couple of choices: one may give out *just* the static public key of the corrupted party, or one may release the complete state of that party. Following [BPR00] we call these the *weak* corruption model and the *strong* corruption model, respectively. As indicated in that paper, forward secrecy is not achievable in the strong-corruption model by two-flow protocols. The difficulty is the following. Let the initiator be denoted by **A** and the responder by **B**. Then if **A** is corrupted after **B** has terminated but before **A** has received the response, then **B** will hold a fresh key but the adversary **Z** can necessarily compute the shared session key K, since the adversary has the exact same information that **A** would have had the it received **B**'s final flow.

2. **No A-to-B authentication.** When the responder **B** accepts a key he has no idea if the flow he is responding to recently came from **A**—it could be years old, for all he knows. This is intrinsic in any two-flow protocol. There is no key-confirmation from initiator-to-responder

3. **No B-to-A authentication.** When the initiator **A** accepts a key following receipt of flow 2, she has no idea if the party from whom she just received a message is the party that was. This property is certainly not intrinsic to 2-flow protocols—but it is intrinsic to any 2-flow protocol where the second flow is independent of the first.

We believe that a well designed AKE protocol would allow for an (optional) third flow that would provide for the properties unachievable by any two-flow protocol. And making flow-2 independent of flow-1 may not be of enough value to offset the loss of property (3).

Summary: **ECMQVS inherits some limitations intrinsic to two-flow AKE protocols.**

6.4 Anonymity attacks

The structure of ECMQVS provides no anonymity in the sense that the the spec indicates that public keys A and B must be exchanged in an authenticated manner. We understand this to mean that the association of **A** to A and of **B** to B must be known and must be checked. Still, a disclosing of identities seems to be a less intrinsic part of ECMQVS than some other protocols, as no digital signatures are directly employed.

Summary: **If supporting anonymity is a goal, the SEC1 ECMQVS definition should be revisited.**

7 Assumptions Underlying Security

Ideally, the security of a key exchange protocol of this style would rely on standard assumptions about the Diffie-Hellman problem.[4] However, standard assumptions such as the computational or decisional Diffie-Hellman assumptions do not appear to suffice to guarantee the security of the ECMQVS protocol. Even assumptions that are less standard but at least have been stated in the literature, such as the Oracle Diffie-Hellman assumption [ABR], do not appear to suffice.

It seems preferable that a standard rely on standard assumptions, a cause of possible concern for ECMQVS. If it does not, we suggest that the documentation of the standard should try to state precisely what assumptions it makes. Here we endeavor to distill some of these assumptions.

We begin by noting an assumption about the hash function $\overline{\ }$ that is necessary for security. We then state a stronger, composite assumption involving the Diffie-Hellman problem and the hash function that is also necessary, and possibly sufficient, for security.

7.1 Group-hash collision assumption

We show that the security of the protocol requires that one make a certain assumption about the hash function $\overline{\ }$ that is not stated in the document and is non-standard. We do not know whether or not this assumption holds for the specific instantiation of $\overline{\ }$ given in the document, but suggest that it is preferable to replace the operation with a hash function for which one can have some argument that the assumption is true.

The assumption. Consider the following computational problem:

Group-Hash Collision Problem: Let G be a generator of a group of size n, and let B be a random group element. The problem is: given B, find $\gamma, \beta \in Z_n$ such that $\gamma \neq 0$ and

$$\overline{\gamma G - \beta B} \quad = \quad \beta \, . \tag{1}$$

The pair (γ, β) will be called a *group-hash collision* for B if Equation (1) holds.

The *group-hash collision assumption* is that this problem is computationally intractable. We will show below that this assumption is necessary for the security of the key-exchange protocol. This raises the question of whether or not this assumption is true for the instantiation of $\overline{\ }$ used in the document. We do not know the answer to this question. The instantiation in question is based on the representation of group elements, which is cause for caution. Replacing the operation with a standard hash function such as SHA-1 would allow one

[4] Ccoupled with an assumption about the hash function H used in computing the session key based on the shared Diffie-Hellman key. We would be content to model this by a random oracle.

to prove the group-hash collision assumption in a random-oracle model, which might be preferable. Meanwhile, the assumption deserves more analysis.

THE ATTACK. We claim that if the adversary can find a group-hash collision in time t then it can compromise the security of the protocol in about the same amount of time. The attack is quite strong. The adversary is assumed to know the public keys of $\mathbf{A}, \mathbf{B}$ and their certificates. (It does not create new public keys.) It does not use a known-key attack, nor does it need to corrupt any entities. Nonetheless it will be able to compute in its entirety —not merely compute partial information about— the session key that an entity $\mathbf{A}$ believes it shares with $\mathbf{B}$.

The attack is as follows. Suppose that entity $\mathbf{A}$ uses static key-pair (A, a) and ephemeral key-pair (X, x). Entity $\mathbf{A}$ sends A and X to $\mathbf{B}$, along with a certificate Cert$\mathbf{A}A$ binding A to $\mathbf{A}$. The adversary $\mathbf{Z}$ captures this flow. It then chooses $\gamma, \beta \in Z_n$ so that Equation (1) holds, where B is $\mathbf{B}$'s public key. It sets $Y = \gamma G - \beta B$. It sends B and Y to $\mathbf{A}$, along with a certificate Cert$\mathbf{B}B$ binding B to $\mathbf{B}$. It then computes

$$S^* \quad = \quad h\gamma(X + \overline{X}A) \; . \tag{2}$$

Note that $\mathbf{Z}$ can compute this because all the terms involved are known to it. Now, the adversary treats S^* as the shared key. (Meaning, if this is not zero, it sets $K^* = H(\texttt{xts}(S^*), \mathit{SharedInfo})$, where *SharedInfo* is the shared information string used in this context.) We will argue below that this is the same shared key that $\mathbf{A}$ will derive. (So that K^* is $\mathbf{A}$'s session key.) This means that $\mathbf{Z}$ has computed the session key of $\mathbf{A}$ with an attack involving just one flow to $\mathbf{A}$. (Entity $\mathbf{B}$, with whom $\mathbf{A}$ thinks it shares the key, has not even entered the picture.)

We now justify the claim that K^* is $\mathbf{A}$'s session key. The shared key computed by $\mathbf{A}$ is by definition

$$\begin{aligned} S &= h(x + a\overline{X})(Y + \overline{Y}B) \\ &= h(x + a\overline{X})((\gamma G - \beta B) + \overline{Y}B) \\ &= h(x + a\overline{X})(\gamma G + (\overline{Y} - \beta)B) \end{aligned}$$

However $\mathbf{Z}$ has chosen $Y = \gamma G - \beta B$ where (γ, β) is a group-hash collision, so Equation (1) implies that $\overline{Y} - \beta = 0$. Hence we get

$$\begin{aligned} S &= h(x + a\overline{X})(\gamma G) \\ &= h\gamma(xG) + h\gamma\overline{X}(aG) \\ &= h\gamma(X + \overline{X}A) \\ &= S^* \; . \end{aligned}$$

The definition of group-hash collision requires $\gamma \neq 0$ so that the shared key is not made 0 by the factor γ above.

7.2 Diffie-Hellman assumptions made

THE ASSUMPTION. Consider the following, novel, computational problem:

Linear Diffie-Hellman (LDH) Problem: Let G be a generator of a group of size n, and let a, x, b be random elements of Z_n. Let $A = aG$, $X = xG$, and $B = bG$. The problem is: given A, X, B, find group elements Y, S such that

$$S \quad = \quad h(x + \overline{X}a)(y + \overline{Y}b)G \; . \tag{3}$$

Here $Y = yG$. The pair (Y, S) will be called a LDH-solution for A, X, B if Equation (3) holds.

The *LDH assumption* is that the LDH problem is computationally intractable. We claim that the LDH assumption is necessary for the security of ECMQVS. (Possibly it is sufficient as well. Whether or not this is true is well worth knowing, but would take considerable work to figure out.) We will explain below why we think this assumption is necessary. Let us first make a few remarks about the assumption.

In the LDH problem, the adversary gets to choose the group element Y. It is this freedom of choice that gives the problem a novel nature as compared to other more standard Diffie-Hellman related problems. The question is whether the freedom to choose Y can facilitate the computation of the corresponding key S.

The assumption involves both the Diffie-Hellman problem and the "hash" function given by $\overline{\ }$. It is easy to see that some simple choices of $\overline{\ }$ make the assumption fail. (For example if $\overline{\ }$ is a constant function returning the value β then the adversary can solve the LDH problem by outputting Y, S with $Y = \gamma G - \beta B$ and $S = h\gamma(X + \overline{X}A)$ for some $\gamma \neq 0$.) We do not know of any attack when the instantiation of $\overline{\ }$ is the one used by ECMQVS, but the problem deserves further study.

We observe that the LDH assumption implies the group-hash collision assumption. (In other words, if the LDH problem is hard, then the group-hash collision problem is hard as well.) So if the LDH assumption is true then the attack of Section 7.1 is not possible.

Ideally, one would like to choose $\overline{\ }$ in such a way that one could prove the LDH assumption based on some standard assumption such as computational or decisional Diffie-Hellman. So far we were not able to find *any* instantiation of $\overline{\ }$, even a random oracle, for which we could provide such a proof. (This does not mean it is not possible, of course.) In the absence of such a proof, the LDH problem ought to receive some direct cryptanalysis before it can be used with confidence as the basis of a standard. We do know the LDH cannot be *harder* than the computational Diffie-Hellman problem in the group generated by G: it is not difficult to show that if the computational Diffie-Hellman problem is easy than so is LDH.

When evaluating the security of a non-standard assumption such as LDH it is highly desirable that the underlying problem have a random self reduction. The existence of such a reduction proves that we do not need to assume the average

case hardness of the problem. Instead, it suffices to assume the problem is hard in the worst case. Most common complexity assumptions, such as CDH, DDH, and Discrete log, have a random self reduction. Fortunately, so does LDH. Given an input (A, X, B) we pick random elements $a', b', c \in \mathbb{Z}_n$ and compute:

$$X' = X + x'G \quad ; \quad A' = \left(\overline{X}/\overline{X'}\right) \cdot A + a'G \quad ; \quad B' = B + b'G$$

Clearly (A', X', B') is a random instance of LDH. Suppose we obtain a solution (Y, S') to this random instance. Then (Y, S) is a solution to the original problem instance, where

$$S = S' - b'\overline{Y} \cdot (X' + \overline{X'}A') - (x' + a' \cdot \overline{X'})(Y + H(Y)B') - b'\overline{Y'}(x' + a' \cdot \overline{X'})G$$

Hence, a solution to a random instance of LDH can be used to solve an arbitrary instance. Thus, it suffices to assume the worst case hardness of LDH (rather than average case hardness).

THE ATTACK. We claim that if the adversary can find a LDH solution in time t then it can compromise the security of the protocol in about the same amount of time. The attack is quite strong. The adversary is assumed to know the public keys of $\mathbf{A}, \mathbf{B}$ and their certificates. (It does not create new public keys.) It does not use a known-key attack, nor does it need to corrupt any entities. Nonetheless it will be able to compute in its entirety —not merely compute partial information about— the session key that an entity $\mathbf{A}$ believes it shares with $\mathbf{B}$.

The attack is as follows. Suppose that entity $\mathbf{A}$ uses static key-pair (A, a) and ephemeral key-pair (X, x). Entity $\mathbf{A}$ sends A and X to $\mathbf{B}$, along with a certificate Cert$\mathbf{A}A$ binding A to $\mathbf{A}$. The adversary $\mathbf{Z}$ captures this flow. It then chooses Y, S so that Equation (3) holds, where B is $\mathbf{B}$'s public key. It sends B and Y to $\mathbf{A}$, along with a certificate Cert$\mathbf{B}B$ binding B to $\mathbf{B}$. But Equation (3) says that S is the session key that $\mathbf{A}$ will compute.

8 Recommended Changes

We list of a few recommended changes to ECMQVS.

1. Throughout the SEC1 document the operator $\overline{P}$ is used to convert a point P on the curve into an integer in a certain range. The operator applies a reduction modulo 2^m to the coordinates of the points P, for some m. The structure preserved by this simple conversion could help an attacker in breaking the complexity assumptions on which ECMQVS relies. We recommend replacing the existing $\overline{P}$ operator with a standard hash function such as SHA-1. This will strengthen the LDH assumption on which the scheme depends.

2. The string *SharedInfo* should include the identities of the entities. They would be listed in a canonical order. Of course doing this would raise the question of how to represent entity names.

3. The trust model should be limited to the static/ephemeral, static/ephemeral one. Only by fixing the trust model can one easily explain what properties one is after. And the other trust models do not seem very desirable for ECMQVS.

Even with these changes ECMQVS should be used with some amount of caution, in view of the lack of proofs and the lack of clarity about what really are the underlying assumptions. See the final discussion, Section 10.

9 Exposition and Minor Corrections

The exposition in SEC 1 is very good but, as we have complained already, the authors are quite vague about the security goals of the key-agreement schemes.

We enumerate some of the typos and other matters that we noticed. Note particularly the comment (p. 11) on octet-string to EC-point conversion.

1. Throughout. The use of *should* in this document should be made consistent with RFC language. Most things that say *should* are obligatory; they should be *must*.

2. Throughout. Numerous missing hyphens. Eg: key-deployment procedure, key-agreement operation, key-derivation function(s),

3. p. 6. a and b have not been introduced at the point when they are used to talk about a Koblitz curve.

4. p. 7. "be a prime field so that" $\rightarrow$ "be a prime field where"

5. p. 11. Step 2.2.2 of the Algorithm in section 2.3.3 is done inefficiently It requires an unnecessary multiplication and inversion. This step can be done more efficiently as follows: If $q = 2^m$ set $\tilde{y}_P = 0$ if $x_P = 0$. Otherwise, let $x_P = z_{m-1}x^{m-1} + \ldots + z_1 x + z_0$ and $y_P = w_{m-1}x^{m-1} + \ldots + w_1 x + w_0$. Let k be the smallest positive integer such that $z_k \neq 0$. Set $\tilde{y}_P = w_k$. This point compression method in $\mathbb{F}_{2^m}$ is much faster and easier to implement than the point compression method currently described in the standard.

6. p. 17, 20, 22, 24, 46, 48. *to receive assurance*, not *to receive an assurance*

7. p. 18, Section 3.1.1.2.1, item 1, "odd" is unnecessary given the rest of this sentence.

8. p. 18, Section 3.1.1.2.1, item 5, maybe you should explicitly allow a probabilistic primality test, as one could conceivably read this as disallowed.

9. p. 18, Section 3.1.1.2.1, item 6. I assume that this formula is right, but I don't get it, and it may be misleading because it shouldn't be necessary to compute a square root of p. Maybe this could suggest a weaker test like $h \in \{2, 3, 4\}$ and $p - \Delta \leq hn \leq p + \Delta$, where $\Delta = 22^\tau + 1$ where $\tau = \lceil \lg p \rceil$.

10. p. 18. x_G and y_G are undefined in the algorithm of 3.1.1.2.1. Add a definition of $(x_G, y_G) = G$.

11. p. 18. bottom of page, step 8: the q should be a p.

12. p. 18. What about the fifth method, that the EC domain parameters are taken from SEC 2! (Well, I guess you could say that this falls under (3), but the statement "in an authentic manner" makes it sound as if some sort of certificate is expected. (Have you signed the SEC 2 document?)

13. p. 19, 21: "random seed" $\rightarrow$ "a specified seed" (to make sure you are not implying that you are checking that the seed is random)

14. p. 27. Personally, I find this $d_{q,U}, Q_{1,U}, d_{2,U}, Q_{2,U}, d_{q,V}, Q_{1,V}, d_{2,V}, Q_{2,V}$ cumbersome and non-mnemonic. (The first thing I do looking at this is to translate into nicer notation.)

15. p. 27. It says that $Q_{1,V}$ should at a minimum be partially valid. But the one place that this primitive is used, in the MQV scheme, requires that $Q_{1,V}$ be (fully) valid. Probably you should require $Q_{1,V}$ to be (fully) valid and $Q_{2,V}$ to be at least partially valid.

16. p. 45. third paragraph – "simultaneously" – there is no requirement for simultaneity, and, in fact, achieving it would be impossible.

17. p. 45. third paragraph – last sentence – this sentence overlooks the possibility of the adversary having subverted the key sharing.

18. p. 47. The second sentence of Section 6.2 is not right; this statement applies to ECDHS, not ECMQVS. ECMQV does not appear to be designed to meet a different set of goals depending on how it is used.

19. p. 48, 6.2.1, item 2, line 1. Delete the words "security level"

20. p. 48, 6.2.2, item 3. The phrase *in an authentic manner* is too vague. Should the point be signed by V—or should it be in a certificate binding V to some entity—or both? This makes a big difference. In particular, does the receiver U believe that entity V and only entity V is associated to $Q_{1,V}$—or might there be other entities in the distributed system associated to V?

    Referring to [LMQSV], one sees that certificates were the intent. But this is not manifest in the SEC 1 document.

21. p. 48, 6.2.2, item 3. The analogous comment.

22. p. 64. "point on E *on* large prime order"

10 Discussion

The state of key-sharing protocols in emerging standards is somewhat disappointing. During the last few years schemes have emerged which pay close attention to efficiency and flexibility, and significant "new" goals, like forward secrecy, have been recognized as important. But nobody has spent the (considerable) time and energy to do first-rate definitions+protocols+proofs targeted for the needs of standards. The result is that nobody knows exactly what a scheme like ECMQVS does, what it is supposed to do, and under what assumptions.

The approach implicit in the SEC 1 key-sharing schemes is the "old" approach of design, attack, and counter-measure. The document effectively forgives protocol problems because of the possibility of effective counter-measures. Effective countermeasures for attacks on ECMQV include making sure to use (static, ephemeral) keys, demanding key-confirmation, putting identities in flows of sessions which get established, and including identities in *SharedInfo*. But we do not buy this way of thinking. One should not have to look outside of what is mandated in a protocol for techniques that will provide the desired assurance guarantees. That is precisely the *job* of the protocol.

The ad hoc nature of ECMQVS does not engender a lot of confidence. Why is the function $\overline{\ }$ the way that it is? Why does this hashing depend on only one of these public keys, and not on two of them, or all four of them? Why is it adequate to have half-length outputs from this hash-function—would not full-length outputs be a more conservative (if slightly less efficient) choice? Looking at the evolution of ECMQVS, an earlier version used a "full-length" hash (redefine $\overline{\ }$) and a derived key of $S = h(\ x + a\overline{A}\ \overline{X}\ )(\ y + b\overline{B}\ \overline{Y}\ )G$. Is there really any reason to think that the modified scheme is superior to the original one, or to possibilities like $S = h(\ x + a\overline{(A, X)}\ )(\ y + b\overline{(B, Y)}\ )G$? Has efficiency been chased *too* much in this case—does half of a multiplication really matter?

We do not wish to conclude too critically. ECMQVS is a good representative for the state-of-the art in key-sharing standards/draft-standards. The attacks on it are not really fatal. But we believe that, if this area is worked, higher-assurance, just-as-efficient schemes will almost certainly emerge.

References

[ABR] M. Abdalla, M. Bellare and P. Rogaway. DHIES: An Encryption Scheme Based on the Diffie-Hellman Problem. *Cryptology ePrint Archive: Report 1999/007*, `http://eprint.iacr.org/1999/007/`.

[BPR00] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. *Advances in Cryptology — Eurocrypt '00.* Springer-Verlag, 2000. Available from `http://www-cse.ucsd.edu/users/mihir`

[Kal98] B. Kaliski. An unknown key-share attack on the MQV key agreement protocol. Manuscript. Proceedings version appeared in RSA

Conference 2000 Europe, Munich, April 2000. Work based on earlier communication to IEEE P1363a and ANSI X9F1 working groups, June 1998.

[LMQSV] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, An efficient protocol for authenticated key agreement. Technical report CORR 98-05, Dept. of C&O, University of Waterloo, Canada, March 1998. Revised August 28, 1998.

[SEC1] Certicom Research, Standard for efficient cryptography, SEC 1: Elliptic curve cryptography. Version 1.0, September 20, 20000. Certicom Corporation. URL: www.secg.org