Evaluation of Security Level of Cryptography

ECDHS (from SEC 1)

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

January 16, 2001

Contents

This document analyzes the method called ECDHS in SEC 1. The analysis is based on the SEC 1 documentation (version 1; 20 September 2000).

1 Summary

The ECDHS (Elliptic Curve Diffie-Hellman Scheme) of SEC 1 is versatile, widely standardized, and builds on well-known techniques. Still, were were not satisfied with it or with its exposition in SEC 1. Our high-level findings are as follows.

1. **Many schemes at once.** The ECDHS is not so much a key-agreement scheme as a key-agreement *schema*—it is a framework which includes various key-agreement schemes. This makes ECDHS very versatile: schemes from this schema are applicable under a wide variety of trust models (that is, assumptions about who has what keys, and what they believe about these keys) and under a wide variety usage scenarios (using static public keys or ephemeral public keys; sending flows signed or not; and so forth). This versatility also makes ECDHS difficult to analyze: the security properties one gets vary enormously according to the particulars, and there are several tens of cases that one could be interested in. The only way to progress appears to be to first extract from the schema the particular set of schemes one is potentially interested in, and then deal with them one-by-one. But there are too many schemes to do this effectively.[1]

2. **Unclear security goals.** The exposition in the SEC 1 spec provides the user with no useful information about what security properties can be expected to follow when a particular scheme is chosen from this schema. Fully understanding the answer to this question would require knowledge beyond anything demonstrated in the spec. We will do our best to clarify this, but our explication will be far from exhaustive or definitive.

3. **Strongest security goals unachieved.** Despite the versatility offered by the ECDHS, no scheme from this schema achieve the strong security properties which are possible and desirable. For example, no scheme from this schema achieve unilateral authentication or mutual authentication; no scheme from this schema is secure in the face of RevealSn queries when either party uses a static key; no scheme is secure in the face of an adversary who may learn coins used for one of the sessions.

4. **Unsafe possibilities.** ECDHS encompasses schemes with both strong and weak security properties. Does the presence of a good scheme from

[1] The diversity we are talking about is not of the sort: *use an arbitrary message authentication code here*, or *use an arbitrary key-derivation function here*; this type of versatility is not problematic. We are speaking of things like: *this key may be static or ephemeral*, or *here you might or might not multiply by h*, or *this flow might or might not be authenticated.* This sort of diversity results in an exposition of fundamentally different schemes all masquerading under a single label.

a schema make the schema right, or does the presence of bad schemes make the schema wrong? We are more inclined toward the latter, particularly because it is so non-obvious what reasonable-sounding instantiations achieve what goals. When, for example, some basic security goal requires a well-informed choice of the optional parameter *SharedInfo*, we would like to regard this as a significant defect: one should not be compelled to make a non-obvious instantiation of an optional parameter in order to achieve basic security goals.

5. **Backwards development.** The deficiencies of ECDHS may stem from a development model that puts the scheme before the goal. Instead of saying: "here are the security goals we are after—how can we best achieve them?" the thinking appears to be more like: "here is a well-known protocol—Diffie-Hellman key-exchange—now how can we adapt it to the EC setting and make a standard of it?" Such an approach is unlikely to give rise to a high-quality standard in a domain as and subtle as this one.

The criticisms may seem harsh. The underlying problem may be, in part, that the state-of-the-art for key distribution is not nearly at the level of the state-of-the-art for other cryptographic goals getting standardized by CRYPTREC (symmetric encryption, asymmetric encryption, digital signatures, and cryptographic hash functions). This is the only domain among those where: (1) there is no complete and agreed-to set of definitions; (2) there are no proven correct protocols for many of the problems one wishes to address; (3) the desired level of variability in security goals, and the round-cost in achieving them, motivates accepting a plurality of schemes; (4) the associated level of definitional complexity makes a mathematical treatment long and weighty; (5) even an informal description of requirements would be the subject of much debate. Thus it would seem to be harder to devise at a good standard for key-distribution than for any other problem considered.

Worse than SEC 1? ECDHS is taken from what we consider to be a nicely executed standard, SEC 1. We gave that standard received a better review than we are giving to this protocol—how is that possible? Because, in the author's opinion, the key-agreement schemes in SEC 1 were the weakest part of SEC 1.

2 The Schema

ECDHS is deceptively simple. Two parties, whom we denote[2] **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, this subgroup having prime order

[2]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.

$n = \frac{|E|}{h}$, where $h \in \{2, 3, 4\}$. Entity **A** has a (secret key, public key) of (a, A), while entity **B** has a (secret key, public key) of (b, B) (here $A = aG$ and $B = bG$). These keys could be static (long-lived) or ephemeral. The parties exchange public keys (possibly in an authenticated way, the standard says). Now:

- *Standard DH method.* Each party checks that the public key received lies in the group $\langle G \rangle$. Assuming it is, entity **A** calculates an EC point $S = aB$, while entity **B** calculates an EC point of $S = bA$. If $S = 0$ the party *rejects*.
- *Cofactor DH method.* Each party checks that the public key received lies in the group E. Assuming it is, entity **A** calculates an EC point $S = haB$, while entity **B** calculates an EC point of $S = hbA$. If $S = 0$ the party *rejects*.

Now the x-coordinate of S is converted into a string s, and the shared session key is $K = H(S, \mathit{SharedInfo})$, where *SharedInfo* is an optional, unspecified string.

3 Goals?

The SEC 1 document states (p. 45):

> *[ECDHS and ECMQV 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.*

Unfortunately, the document never explains what security goals is intended to follow if the scheme is applied in some particular way. In fact, neither the specific security goals nor the scope of how it might be applied is discussed at a level allowing a user to understand what security goal will follow from what way to use ECDHS. This is a problem, making it hard for a user to know how or if to use ECDHS.

Two of the goals mentioned above are given descriptions:

implicit key authentication (p. 52): only authorized parties are possibly capable of computing any session keys.

forward secrecy (p. 52): ... a session key established between some parties will not be compromised by the compromise of some of the parties' static keys in the future.

We do not find descriptions of **unilateral authentication** or **mutual authentication** in the document, though other sources give descriptions at a similar level of refinement.

The above descriptions don't go far to clarify the key-agreement goals. What is the intended attack model? For example, can session keys be revealed? Can

private coins? Can parties be corrupted? What does one get when doing so? Is computing some bits of the session key ok—just not all of it? And so forth.

Despite considerable research in this area, it is a still-unresolved question as to what exactly are the goals that a protocol like ECDHS *can* achieve, which are the important goals *to* achieve, and what goals are actually that a protocol like this *does* achieve under different scenarios.

Trying to have a clearer description of the intended goals is not just of "academic" interest. When Kaliski found an "unknown key-share attack" on the MQV protocol [K98] of SEC 1, it was unclear if this amounted to a successful attack. It was unclear because the security goals of the scheme had not enunciated carefully enough.

For us, the "basic" goal of a KD scheme can be stated (still quite informally) as follows:

> *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 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 in some ways a higher standard than the goals set out for ECDHS.

4 From Schema to Schemes

To get an understanding of the security characteristics of ECDHS one needs to extract from this schema some particular schemes to look at.[3] From the SEC 1 spec it is not immediately obvious what are the relevant "dimensions" by which the schemes differ. We single out what seem to be the principal ways in which the schemes differ. Refer to Figure 1.

STATIC VS. EEPHEMERAL. A central characteristic of the scheme is whether the key pairs are static or ephemeral:

1. A (public key, secret key) is **static** (long-lived) if it will be used for multiple sessions.

2. A (public key, secret key) is **ephemeral** if it will be used for just one session.

We will also use these words to apply to just the public key or its secret key.

The above characteristics separately apply to the initiator **A** and the responder **B**. Thus there are four possibilities: ordering by initiator/responder, we denote these four cases by static/static, static/ephemeral, ephemeral/static,

[3] Again we are using language somewhat at odds with SEC 1. ECDHS is one scheme in SEC 1. We are saying that, in analysis, one can not think of it that way.

A	B	Scheme
static/ephemeral	static/ephemeral	
nocert/cert/strongcert	nocert/cert/strongcert	
unsigned/signed	unsigned/signed	standard/cofactor

Figure 1: *Characteristics on which to differentiate schemes from the ECDH schema. These 192 possibilities still do not capture some security-relevant considerations, as discussed in the text.*

and ephemeral/ephemeral. Similar notation will be used for other scheme attributes which separately apply to the initiator and the responder.

NOCERT VS. CERT VS. STRONGCERT. One can assume that:

1. **nocert.** The receiver of the public key does not know the agent to whom this key is bound (e.g., he doesn't have any sort of certificate binding this public key to a named entity).

2. **cert.** The receiver of the public key knows that it belongs to a particular named entity (e.g., he has a certificate establishing this binding, and the receiver of the PK trusts the CA that asserts this binding).

3. **strongcert.** The receiver of the public key knows that it belongs to a particular named entity, and the receiver knows that that entity has the corresponding secret key. For example, the CA who provided the certificate might have required the entity requesting it to prove knowledge of the secret key.

As we shall see, the level of assurance provided by "cert" does not seem to be very useful when certificates are used in ECDHS. The problem is that an adversary **Z** can copy a principal's public key and have it certified (in the cert-sense) as her own. On the flip side of this, "strongcert" would provide the needed guarantees—but we would be weary of a protocol that made such a strong assumption on what a certificate implies. A good protocol would (and could) achieve its goals assuming the lower, cert-level of guarantee.

The use of static public keys that are not known (by certificates or other means) to be associated to some particular principal might not be so interesting. Similarly, the use of ephemeral public keys that have corresponding certificates is an unlikely possibility (I have never heard of getting a certificate for an ephemeral PK). Still, logically, the presence or absence of a known entity-PK binding is orthogonal to the key being static or ephemeral.

UNSIGNED VS. SIGNED. When public keys (static or ephemeral) are exchanged, they may be exchanged in an unauthenticated or an authenticated way. The latter would normally be accomplished by signing the public keys. We shall distinguish this dimension as **unsigned/signed**. The latter, when applied to a

party **U**, means that the *other* party **V** is certain of the originator of that flow. As with nocert/cert/strongcert, the intent is not to mandate the mechanism, but to suggest the use of an abstracted model which corresponds to the assurance one would normally obtain through the mechanism of signatures.

STANDARD VS. COFACTOR. The ECDHS comes in a **standard** version and a **cofactor** version (where one multiplies by h). These are significantly different protocols.

FURTHER DIMENSIONS. The ECDHS allows an optional string *SharedInfo* to be used in the key-derivation process. One needs to discuss security according to what that string may be. In the "worst" case the string is empty or security-irrelevant. (It is conceivable, though perhaps a bit far-fetched, that reasonable use of this string could compromise security.) However, there are better choices for this string (as hinted at in the SEC 1, Appendix B) where one can salvage some security properties that would otherwise be absent. Since this string is unspecified and security-relevant, there are now infinitely many protocols to try to analyze. We will discuss some uses of *SharedInfo* in the sequel.

TAXONOMY. Referring to Figure 1, reading left-to-right and then down, we have singled out 192 possibilities: the initiator **A** may use a static or ephemeral public key; the responder **B** may use a static or ephemeral public key; the initiator's public key might not or might be known by the responder to be associated to the initiator's identity—and, in the latter case, the responder might not or might take for granted that the initiator is the one that originated that public key; the responder's public key might not or might be known by the initiator to be associated to the responder's identity—and, in the latter case, the initiator might not or might take for granted that the responder is the one that originated that public key; the public key (purportedly) provided by the initiator might not or might be known by the responder to have been sent by the initiator; the public key (purportedly) provided by the responder might not or might be known by the initiator to have been sent by the responder; the scheme itself may use the standard key-derivation method or the cofactor key-derivation method. As indicated above, this taxonomy of schemes still omits the role of *SharedInfo*.

A STAGGERING NUMBER OF POSSIBILITIES. It would seem nearly impossible to fully understand what is and is not achieved under all of these possibilities. But the situation is perhaps not completely hopeless. As indicated, some of the possibilities are not interesting. Some of the added elements don't add much. In looking at attacks, some of the variability will be irrelevant. Still, the above discussion should already make clear that there are major obstacles to fully analyzing a schema of this ilk. To our knowledge, it has never been done. We suggest that a schema specification like this one is not amenable to careful analysis by cryptographers simply because it is so open-ended in potentially relevant directions as to admit several tens of fundamentally different schemes—more than anyone could carefully deal with.

5 Efficiency

ECDHS is good in terms of efficiency. The computational work associated to ECDHS is well-approximated by looking at the number of multiplications in the elliptic-curve group that each party must perform, the number of signature generations, and the number of signature verifications.

For the static/static cases, where the associated key for the pair of principals has been cached, one needs **zero** multiplications, signature generations, or signature verifications. This is the case that would be associated to fast rekeying, and no algebraic/public-key operations would be performed.

For the (ephemeral/ephemeral, nocert/nocert, unsigned/unsigned, cofactor) case we each party needs **two** multiplication (and no signature operations). One of these can be performed in advance, off-line.

For what we regard as the "best" cases, namely (ephemeral/ephemeral, signed/signed, cofactor), each party needs **two** multiplications, but each party must also generate one signature and verify one signature. One of the multiplications, as well as the signature generation, can be done in advance, off-line.

For all cases besides the cached static/static ones, the corresponding standard (non-cofactor) scheme adds one more multiplication.

Of course it is not only the number of multiplications that matters, but (among other factors) the size of the group where we are doing these multiplications. As we shall see, most of the schemes do not support strong provable-security results, and so it is impossible to get an understanding of the appropriate parameter choices by looking at the strength of the reductions involved. Nonetheless, a security level of 80 bits, that is $n = 160$, is probably adequate in practice.

6 Representative Attacks

We give some representative attacks on various schemes from this schema. None of the attacks are deep, and we assume that most or all of them are known.

6.1 Fixed-session-key attack (static/static)

We consider the case in which the DH key exchange is based on a pair of static public keys: entity **B** has the public key A of entity **A** (where $A = aG$). Similarly, entity **A** has the public key B of entity **B** (where $B = bG$). If **A** and **B** are already in this state, no flows are needed. If either or both are not yet in this state, then likely one or two certificates is flowed, either from the entity in question or from a registry service. The entities would be assumed to have the public key for the CA. Though it is irrelevant for us now, one might assume that **B** knows this key to be associated to **A** (eg., we are in the cert case) and **A** knows this key to be associated to **B**. Pictorially, the (static/static, cert/cert, unsigned/unsigned, standard) protocol might be realized by:

$\mathbf{A}^{(a,A),\ \mathbf{B}:B}$ $\mathbf{B}^{(b,B),\ \mathbf{A}:A}$

PID = **B**	PID = **A**
$S = aB$	$S = bA$
$K = H(S)$	$K = H(S)$

Here the superscripts for the parties serve as a shorthand for the trust model: they indicate what a party "knows." The notation $\mathbf{U} : U$ is used for the cert model, indicating that the party "knows" that **U** has been associated to public key U.

The static/static trust model is useful. (This is the model of SKIP.) If entities **A** and **B** can be regarded as already being in possession of one another's public keys, then they *implicitly* have a shared key K based on these two public keys. However, the key K is *not* a session key in any sense. It does not vary from run to run and must not be used as a session key. Consider, for example, what happens if K is used in counter-mode encryption. Disaster! The same Vernam pad is used in every session, completely sacrificing privacy. If the keys used in ECDHS are both long-lived, then the protocol does not accomplish the minimum requirements of properly distributing a session key: getting in the hands of the participants a new key with each session.

Distributing a proper session key. There is some way out, via the optional string *SharedInfo*. If *SharedInfo* is something that is guaranteed to never be reused across sessions, then some measure of session-key security is possible when both parties use static public keys. Consider, for example, the following scenario, where α and β are random strings selected by **A** and **B**, respectively:

$\mathbf{A}^{(a,A),\ \mathbf{B}:B}$	$\xrightarrow{\quad\alpha\quad}$	$\mathbf{B}^{(b,B),\ \mathbf{A}:A}$
	$\xleftarrow{\quad\beta\quad}$	
$S = aB$		$S = bA$
$\mathit{SharedInfo} = \mathbf{A} \parallel \mathbf{B} \parallel \alpha \parallel \beta$		$\mathit{SharedInfo} = \mathbf{A} \parallel \mathbf{B} \parallel \alpha \parallel \beta$
PID = **B**		PID = **A**
$K = H(S,\ \mathit{SharedInfo})$		$K = H(S,\ \mathit{SharedInfo})$

Now the protocol distributes a fresh session key each time it is run. In the envisaged scenario where **A** and **B** are already in possession of the certificates, the flows are only an exchange of α and β.

The protocol does **not** achieve forward secrecy: if either a or b should at some point become known, all previous session keys will be compromised.

The protocol is secure against strong reveal queries (which give the adversary the coins of this session) for the simple reason that there are no private random coins used with a session.

Is it acceptable to eliminate the value β from being flowed, setting *SharedInfo* = α? No; the adversary who obtains one session key can now initiate a new session (playing the role of **A**) which uses this session key.

Is it acceptable to eliminate the value α from being flowed, setting *SharedInfo* = α? No; the adversary who obtains one session key can now initiate a new session (playing the role of **B**) which uses this session key.

If the entities have approximately synchronized clocks than one alternative is to make *SharedInfo* a time-varying value provided by the **A**. Entity **B** would have to check that this value is timely, and entity would have to reject if the same value is presented twice.

What has been said in this section holds true for other static/static settings.

Summary: **Without using *SharedInfo*, the static/static ECDHS is insecure in the most fundamental way: it does not produce a session key. But it does provide a shared static key. The protocol can be made to distribute a session key if *SharedInfo* incorporates nonces provided by both parties. Even then, forward secrecy is not obtained.**

The role of **A** and **B**, which we also inserted into *SharedInfo*, will be made clear in the next subsection.

6.2 Unknown key-share attacks

The static/static cases of ECDHS has another problem—one quite endemic to the ECDH schemes. Let us continue to assume the cert/cert case but think, concretely, that the binding of an entity **U** to his public key U is provided by a certificate $\mathrm{Sign}_{\mathrm{ca}}(\mathbf{U} \leftrightarrow U)$ obtained from entity **CA** (with widely-known public key CA, and well-secured secret key ca). Let us assume that **U** can obtain a certificate bound to **U** simply by demonstrating that he is **U** and presenting the desired public key U. This is typical. It is the "cert" setting.

An adversary **Z** takes the public key A of **A** and gets a certificate $\mathrm{Sign}_{\mathrm{ca}}(\mathbf{Z} \leftrightarrow A)$ saying that A is **Z**'s public key. Similarly, **Z** takes the public key B of **B** and gets a certificate $\mathrm{Sign}_{\mathrm{ca}}(\mathbf{Z} \leftrightarrow B)$ saying that B is **Z**'s public key. Now entity **A** engages **Z** in a conversation where **A** assumes a binding of $\mathbf{Z} \leftrightarrow B$ (obtained by **Z** providing **A** with such a certificate). The adversary **Z** engages entity **B** in a conversation where **B** assumes a binding of $\mathbf{Z} \leftrightarrow A$ (obtained by **Z** providing **B** with such a certificate). The adversary simply relays the messages between **A** and **B**. In this way, **A** establishes a shared session key with **B**, but both entities believe, wrongly, that they are speaking to entity **Z**. Pictorially:

$$\mathbf{A}^{(a,A),\,\mathbf{Z}:B} \quad \longleftrightarrow \quad \mathbf{Z} \quad \longleftrightarrow \quad \mathbf{B}^{(b,B),\,\mathbf{Z}:A}$$

$$\begin{array}{c} \mathrm{PID} = \mathbf{Z} \\ S = bA \\ K = H(S) \end{array} \qquad\qquad \begin{array}{c} \mathrm{PID} = \mathbf{Z} \\ S = bA \\ K = H(S) \end{array}$$

This SEC 1 document refers to this type of problem as an "unknown key-share attack."

Above, PID is the "partner ID"—the entity that the indicated principal believes it has just spoken to. We regard it as an important characteristic of a KE protocol that ones partner is whom one believes it to be. For us, this is a core aspect of a KD protocol, not an optional attribute.

One can similarly have only one of the two parties believe, wrongly, in the identity of its partner.

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

Summary: **The static/static ECDHS insecure in the sense of being susceptible to unknown key-share attack. To overcome this, include identities of protocol participants in *SharedInfo* (along with nonces of protocol participants).**

Strong certificates. Another possibility is to require an entity **U** who wants a certificate $\mathrm{Sign}_{\mathrm{ca}}(\mathbf{U} \leftrightarrow U)$ to "prove" that he is in possession of the secret key corresponding to U. Implementing this latter approach is problematic, and it probably should not be relied on. In particular, if the CA asks a party presenting a PK U to sign a random challenge, or to decrypt a random string, there is no guarantee that **U** will not be able to accomplish this by again using the protocols deployed within the distributed system. That is, "ordinary" proofs of knowledge are not adequate to realize the strongcert model.

For the strongcert model, even the "economic" incentive is problematic. Why should I obtain a strongcert certificate when it is not I who benefits from it, but others who use my certificate? Are we to charge *them*? That would be antithetical to creating a widely available PK infrastructure.

The strongcert model can be avoided: secure KD protocols can be constructed under cert model, and we feel strongly that reaching beyond the cert model is expecting too much of the computing infrastructure.

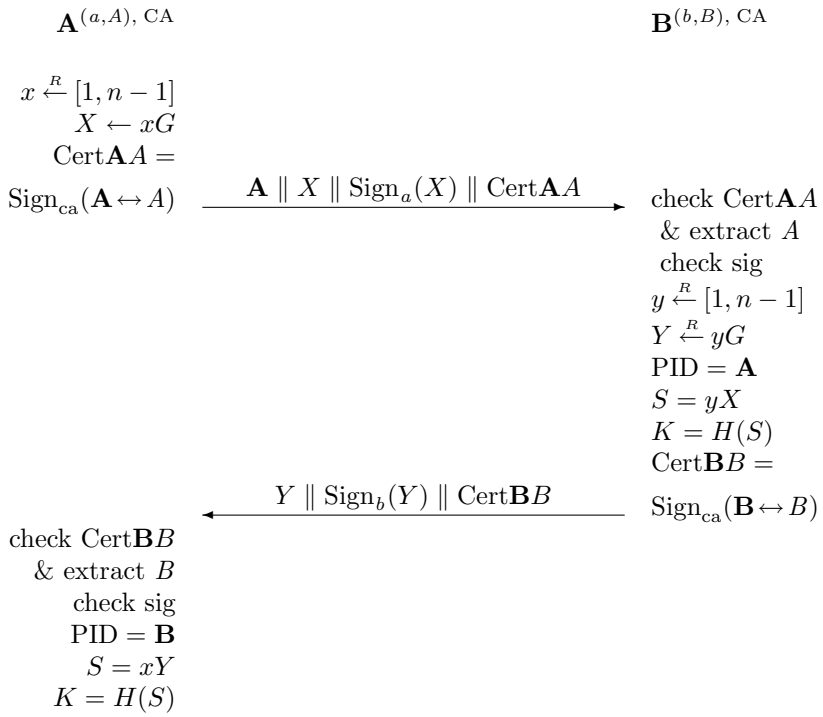
Summary: **For practical and useful protocols, the strongcert model should not be assumed.**
Henceforth we will ignore strongcert.

More unknown key-share attacks What has just been demonstrated for the static/static case actually has nothing to do with whether keys are static or ephemeral: it is applicable in all scenarios—including the signed/signed cases, where each entity knows that the public key received from a party **U** really did originate with **U**. It is possible that the designers of ECDHS assumed otherwise, suggesting that exchanging public key in an authentic manner would improve upon some (unspecified) security goals:

> [the security goals provided by ECDHS depend] on issues like whether or not public keys are exchanged in an authentic manner . . .[SEC 1,

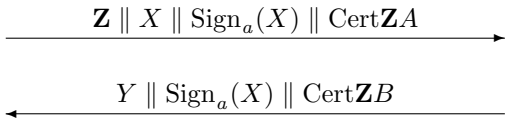
p. 45]

Let us examine this, by considering (ephemeral/ephemeral, nocert/nocert, signed/signed, standard) protocol. It might be realized as follows:



(Note that the presence of the certificate does not make this the cert/cert case, which would mean that the ephemeral keys X and Y had certificates. Here the certificate is only used to allow the signature to be checked.)

This is no better than before. The adversary $\mathbf{Z}$ records X and $\mathbf{A}$'s signature of it, and Y and $\mathbf{B}$'s signature of it. The adversary $\mathbf{Z}$ goes to $\mathbf{CA}$ and obtains a certificate $\text{Cert}\mathbf{Z}A = \text{Sign}_{\text{ca}}(\mathbf{Z} \leftrightarrow A)$ and a certificate $\text{Cert}\mathbf{Z}B = \text{Sign}_{\text{ca}}(\mathbf{Z} \leftrightarrow B)$. Now the adversary replaces actual certificates with the ones she has obtained, but keeps the old signatures:



In short, authenticating the public keys that are exchanged (signing them) does nothing to address the problem because the signatures only bind the exchanged public keys to other public keys, and in the nocert and cert models the adversary already had the ability to bind these public keys to other entities. As

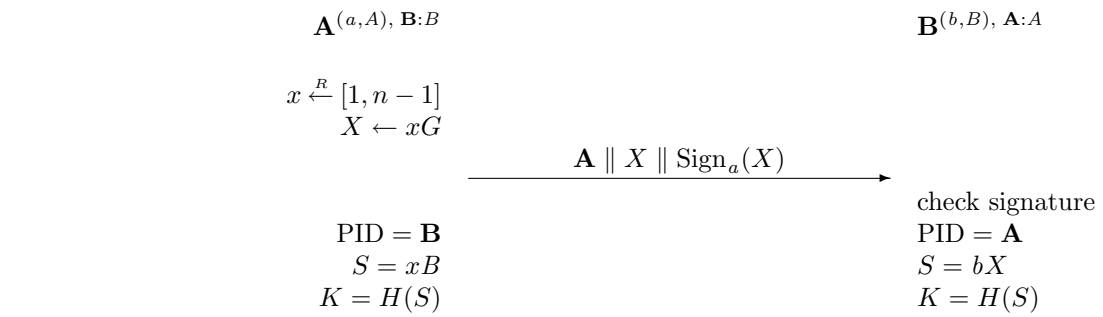
before, one would have to go to the (rather problematic) strongcert model, or (better) to add entity names to *SharedInfo*, as we did before.

Summary: **All versions of ECDHS (excluding the strongcert model) are susceptible to unknown key-share attack. The problem can be addressed by including identities in *SharedInfo*.**

6.3 Captured session-key attacks (at-least-one static key)

All of the static/static, static/ephemeral, and ephemeral/static cases, regardless of certificates or signatures, fail rather badly as soon as we allow for RevealSn queries. That is, if the adversary can obtain some old session key, then the adversary can win. Let is illustrate.

Suppose, for concreteness, the following ephemeral/static scenario:



Now suppose that the adversary **Z** can obtain a session key K and has recorded the corresponding flow from **A** to **B** that gave rise to it. Then the adversary can, from that point on, get **B** to have a shared key with **Z** which **B** believes to be shared with **A**. The adversary simply repeats the contents of **A**'s earlier flow. That flow determines the (already known) session key K.

Adding identities to H is irrelevant. Going to the strongcert The protocol is fundamentally insecure against RevealSn queries. We conclude:

Summary: **If either party uses a static key, then loss of a single session key destroys future security.**

6.4 Key-translate attacks

Here is an attack which many will not regard as a "real" attack.

Assume the standard (as opposed to cofactor) version and, for concreteness, the (ephemeral/ephemeral, nocert/nocert, unsigned/unsigned) case. Suppose that **A** sends **B** the public key X and the adversary **Z** replaces it by αX, for some $1 < \alpha < n$. When **B** sends back a public key Y the adversary replaces it by αY. Then both parties will compute the same shared key $S = \alpha yX = \alpha xY = \alpha xyG$, which will be converted into the session key K. Is this a damaging attack?

The SEC 1 authors realized that this would be a damaging attack if α could be chosen so as to force $S = \alpha xyG$ to be 0, or to be one of some small set of values. Thus they insist that each party rejects a value of $S = 0$, and they require each party to check that the public key U which is received from the other party satisfies $nU = 0$. But if all the adversary **Z** accomplished was to shift the value S from which the session key is derived from its (unknown) value K to its (unknown) value αK, does it matter?

Conceivably. In some treatments of session-key distribution there is the idea of a session ID, SID, that should uniquely name each session. The session ID can be useful in protocols; for example, it can be the basis of auditing. The session ID should be a common, publicly-knowable value that the communicating peers hold.

The usual way to define the SID would be as the concatenation of the flows. But this method *does not work* in the face of the "key translate" attack we have shown. One party will have an SID of $(X, \alpha Y)$, while the other party will have an SID of $(\alpha X, Y)$. Since they have different session IDs they will not be considered to be "partners" in their communication. Consequently, in the formal definitions, it will be fine to RevealSn the session key of one of these entities and it should not compromise the session key of the other. But of course it would compromise the session key of the other: they parties hold the same session keys. From our outside point of view, the parties are partners, despite their view of the flows being different.

Maybe one could have defined the SID in some other manner—but all the alternatives seem to be problematic. Just viewing the contents of all flows it will be information-theoretically impossible to figure out who is partnered with whom. That is, if the adversary **Z** gets a frist message from 100 instances of **A**, randomly translates each of them, and then sends the translates to 100 instances of **B**, we will have no way to ascertain who is partnered with whom. It is only in the adversary's head.

One could try to use the session key itself as the basis of partnering. This too is problematic. Suppose we have a strong reveal query, where S itself will be manifest to the adversary. Then the adversary α-translates the flow from **A** to **B**. The two entities get different keys, and so they are not partnered with one another. Doing the strong-reveal query of one party is not supposed to compromise the other. But it does, since the two pre-session keys (the S-values) differ by an adversarially-known constant α.

Another way to describe the defect is to go back to the intuition of early work in authentication, Bellare-Rogaway 93, where it was said that the most that an adversary should be able to do is to act like a (possibly broken) wire. Here, in the session-key translation attacks, the adversary is able to depart from that abstraction with impunity.

The attack of this subsection is not a strong, practical attack—many would say, with justification, that it is a non-attack. But it illustrate that desirable, clean, definitions are not being achieved.

6.5 Reveal$ attacks

When formalizing security one may wish to allow a Reveal$ query that lets the adversary obtain the internal coins of an instance of a protocol. There are two versions: one in which a principal may erase his coins, and one in which he may not. The distinction will not be important for us.

A Reveal$ query models something quite different from the RevealSn query that has been introduced in earlier works. The latter captures the possibility that a session key may be leaked (even intentionally) by the protocol the authenticated key exchange was furnishing a session key for. In contrast, a Reveal$ query models the possibility that the computing environment on which the instance ran used an inadequate source of randomness, or got compromised. Suppose, for example, that a user authenticates from a public work station. When the user is done using this workstation it might be possible for an adversary to come in and obtain the secret coins that had previously been used. Alternatively, the machine may have been hacked beforehand so as to issue predictable coins. Alternatively the source of randomness on the public workstation might depend on a broken system clock, or on non-existent network traffic.

In general, we would like that compromise of the coins of an instance compromises only that instance. But this is not the case with ECDHS. Consider, for concreteness, the (ephemeral/ephemeral, nocert/nocert, signed/signed, standard) scheme. Let $\alpha = X \parallel \mathrm{Sign}_a(A)$ be the first flow (it is sent by **A**). If the adversary **Z** *ever* learns the secret coins a associated to $A = aG$ then she can establish new sessions with **B** wherein she knows the secret key. All the adversary **Z** has to do is to replay the flow α, get back the response β, and, by assumption, she now has everything needed to compute the session key K.

This is a deficiency roughly comparable to disrupting the forward secrecy: loss of something that *shouldn't* be leaked but could, possibly, be leaked, is having consequences worse than what is possible and desirable.

Summary: **ECDHS strongly assumes that secret coins used by authenticating parties will remain secret forever.**

6.6 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.

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

6.7 Annonymity attacks

The structure of ECDHS provides no anonymity. When used in a "strong" mode, like (static/static, signed/signed), the presence of the digital signatures make identities manifest. There may be environments wanting authenticated key exchange where some level of annonymity is also a goal.

Summary: **If supporting anonymity is a goal, ECDHS is inappropriate.**

7 Positive Results

7.1 A good case for ECDHS

For getting good security results we consider the scheme depicted in Figure 2. Here we assume the (ephemeral/ephemeral, nocert/nocert, signed/signed, standard) case. We have also instantiated the optional string *SharedInfo* as the concatenation of the identities of the parties. (The order is important.) We have made the trust assumptions provided by any certificates implicit, so certificates are not flowed in the protocol. We believe that this protocol meets a reasonable notion of security under reasonable assumptions and that this can be proven. In this section we will state our claim more precisely, and then provide some justification for this claim.

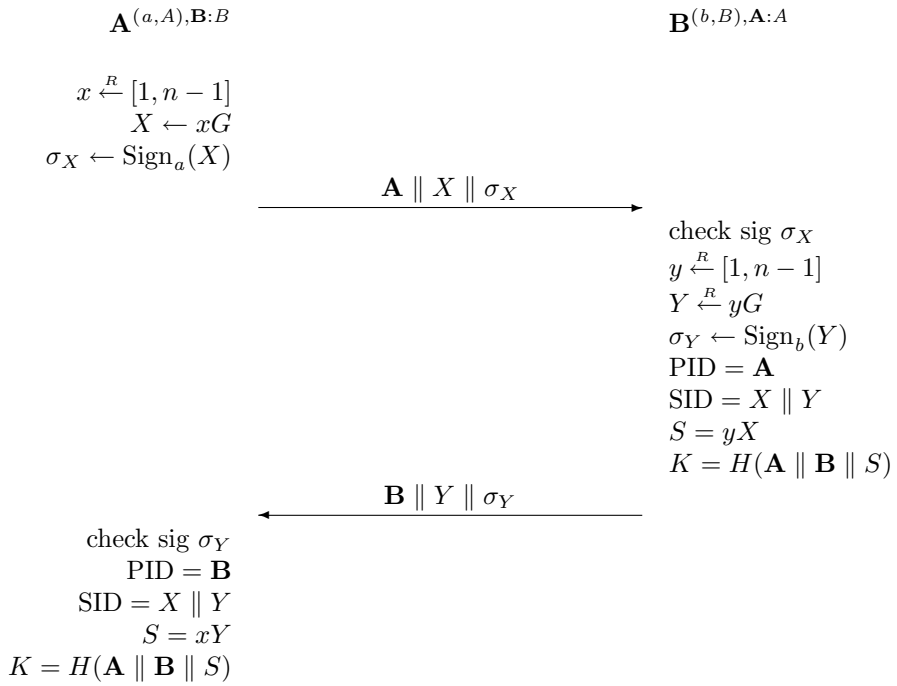


Figure 2: *Protocol P. Each entity is assumed to already be in possession of an authentic copy of the public key of any other entity with which it communicates.*

We will prove security under the computational Diffie-Hellman (CDH) assumption and the assumption that the signature scheme being used is secure. We begin by outlining these assumptions more precisely.

CDH. Let $\mathcal{G}$ denote the group over which we work, with G denoting a generator and n the group size. We assume some fixed representation for group elements, and implicitly switch between group elements and their string representations. A DH-adversary D gets as input two random group elements $\overline{x}G$ and $\overline{y}G$ which we call the *challenge points*. The adversary outputs a list of group elements, $z_1, \ldots, z_q$, and we say that it wins if this list contains the Diffie-Hellman key $\overline{xy}G$ associated to the challenge points. We define

$$\begin{aligned} \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G}}(D) &= \Pr[\overline{x}, \overline{y} \leftarrow \{1, \ldots, n\} : \ \overline{xy}G \in D(\overline{x}G, \overline{y}G)], \text{ and} \\ \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G}}(t, q) &= \max_{D}\{\ \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G}}(D)\ \}\ , \end{aligned}$$

where the maximum is over all adversaries that run in time at most t and output a list of q group elements. Here t includes the description size of adversary D.

Signatures. We let $SIG = (\mathrm{SigKG}, \mathrm{Sign}, \mathrm{Vf})$ denote the signature scheme

in use. (Assume for simplicity that all entities use the same scheme.) The description of the scheme includes: a (randomized) key-generation algorithm SigKG which produces a secret signing key sk and matching public verification key pk; a signing algorithm Sign that takes sk and a message M to produce a signature $\sigma = \mathrm{Sign}_{sk}(M)$; a verification algorithm Vf that takes pk, a message M and a candidate signature σ to output a bit. We say that σ is a valid signature of M under pk if $\mathrm{Vf}_{pk}(M, \sigma) = 1$. Obviously, the signing algorithm produces valid signatures. The scheme is assumed to withstand forgery under chosen-message attack [GMR84], concretized as follows. A signing adversary (also called a forger) F that attempts to break the signature scheme takes as input a public key pk, and is given access to an oracle $\mathrm{Sign}_{sk}(\cdot)$ for producing signatures under the corresponding secret key. It wins if it outputs a message-signature pair in which the message is new (meaning, was not a query to the signing oracle) and the signature is valid. We let $\mathsf{Adv}^{\mathrm{sig}}_{SIG}(F)$ denote the probability that F wins, taken over the choice of keys, coins of the signing oracle, and coins of F. We let $\mathsf{Adv}^{\mathrm{sig}}_{SIG}(t, q)$ denote the maximum, over all F having running time at most t and making at most q sign-oracle queries, of $\mathsf{Adv}^{\mathrm{sig}}_{SIG}(F)$.

KD SECURITY. We use the syntax, model and definitions of security of [BPR00], appropriately modified and restricted. Specifically, we consider an asymmetric setting rather than the symmetric setting they consider; we do not divide the entities into clients and servers, but view them alike; and we are not concerned with dictionary attacks. Let us now briefly overview and detail the main elements of our setup.

The scheme is specified via the protocol P summarized in Figure 2, and an associated long-lived-key generator LL responsible for producing the static keys used by the parties. In this case the long-lived key generator simply gives each entity a secret signing key and matching public verification key, these pairs being independently generated according to the algorithm SigKG. Associated to an adversary $\mathbf{Z}$ attacking the scheme is an experiment measuring its success. We recall that in model of [BPR00] the adversary is viewed as interacting with oracles of the form $\Pi^i_{\mathbf{U}}$, representing instance i of entity $\mathbf{U}$, and communicates with them via oracle queries. The types of queries allowed and their semantics determines the kinds of attacks being captured. We will allow the following subset of the queries defined in [BPR00]:

- $\mathsf{Send}\,(\mathbf{U},\ i,\ M)$ — This sends message M to oracle $\Pi^i_{\mathbf{U}}$. The oracle computes what the protocol says to, and sends back the response. Should the oracle accept, this fact, as well as the SID (session ID) and PID (partner ID), will be made visible to the adversary. Should the oracle terminate, this too will be made visible to the adversary. To initiate the protocol with initiator $\mathbf{A}$ trying to enter into an exchange with responder $\mathbf{B}$ the adversary should send message $M = \mathbf{B}$ to an unused instance of $\mathbf{A}$. A Send-query models the real-world possibility of an adversary $\mathbf{Z}$ causing an instance to come into existence, for that instance to receive communications fabricated by $\mathbf{Z}$, and for that instance to respond in the manner prescribed by the

protocol.

- $\mathsf{RevealSn}$ $(\mathbf{U},\ i)$ — If oracle $\Pi^i_{\mathbf{U}}$ has accepted, holding some session key K, then this query returns K to the adversary. This query models the idea (going back to Denning and Sacco [DS81]) that loss of a session key shouldn't be damaging to other sessions. A session key might be lost for a variety of reasons, including hacking, cryptanalysis, and the prescribed-release of that session key when the session is torn down.
- $\mathsf{Corrupt}$ $(\mathbf{U}, U)$ — The adversary obtains the secret key of entity $\mathbf{U}$, and also replaces the public key of $\mathbf{U}$ by a value U of its choice. (We adopt what [BPR00] called the *weak corruption model*, meaning the adversary does not get the internal state of active instances. As discussed already and in [BPR00], forward secrecy for two-flow protocols is impossible in the stronger model.) This query models the possibility of subverting a principal by, for example, witnessing a user type in his password, installing a "Trojan horse" on his machine, or hacking into a machine. Obviously this is a very damaging type of query. Allowing it lets us deal with forward secrecy and the extent of damage which can be done by breaking into a server.
- Test $(\mathbf{U},\ i)$ — If $\Pi^i_{\mathbf{U}}$ has accepted, holding a session key K, then the following happens. A coin b is flipped. If it lands $b = 0$, then K is returned to the adversary. If it lands $b = 1$, then a random session key, drawn from the distribution from which session keys are supposed to be drawn, is returned. This type of query is only used to measure adversarial success—it does not correspond to any actual adversarial ability. You should think of the adversary asking this query just once.
- Oracle (M) — Finally, we give the adversary oracle access to the function H used in key-derivation. It is selected at random from the set of all functions mapping some appropriate domain to $\{0,1\}^k$. Here k is the desired length of the session key, and the domain is some finite set that includes all strings of the form $\mathbf{A} \parallel \mathbf{B} \parallel S$ where $\mathbf{A}, \mathbf{B}$ are taken from the (finite) set of possible identities and S is taken from the group. In particular, we are in the random oracle model [BR93].

The notion of security we target, from [BPR00], is that of authenticated key exchange with (weak) forward secrecy. As per this definition, the adversary wins if the oracle to which it issues its Test query has terminated, the session key of this oracle is *fs-fresh*, and the adversary correctly guesses the value of the challenge bit b associated to the query. The advantage $\mathsf{Adv}^{\text{ake-wfs}}_{P,LL}(\mathbf{Z})$ of the adversary is obtained by doubling the winning probability, and then subtracting one. Fs-freshness is defined based on a partnering function determined by session ids, and we refer to [BPR00] for the details of this notion as well as the notion of termination. We are interested in the maximal advantage of an adversary as a function of her resources. The adversary resources of interest are:

- t — the adversary's running time. By convention, this includes the amount of space it takes to describe the adversary.
- $q_{\rm se}$, $q_{\rm re}$, $q_{\rm co}$, $q_{\rm H}$ — these count the number of Send, RevealSn, Corrupt, and Oracle queries, respectively.
- $q_{\rm en}$ — the number of different entities that are "active," meaning an oracle associated to this entity has received some query, or a flow claiming to be from this entity has been sent.

When we write $\mathsf{Adv}^{\text{ake-wfs}}_{P,LL}(\textit{resources})$, overloading the Adv-notation, it means the maximal possible value of $\mathsf{Adv}^{\text{ake-wfs}}_{P,LL}(\mathbf{Z})$ among all adversaries that expend at most the specified resources.

Security claim. The claim we make is the following.

Theorem 1 *Let $\mathcal{G}$ be a group and SIG a signature scheme. Let P be the key exchange protocol of Figure 2 over these primitives, and LL the associated long-lived-key generator. Then*

$$\mathsf{Adv}^{\text{ake-wfs}}_{P,LL}(t, q_{\rm se}, q_{\rm en}, q_{\rm H}) \;\leq\; q_{\rm se}^2 \cdot \mathsf{Adv}^{\rm cdh}_{\mathcal{G}}(t, q_{\rm H}) + 2q_{\rm en} \cdot \mathsf{Adv}^{\rm sig}_{SIG}(t, q_{\rm se}) \;.\; \blacksquare$$

In other words, the protocol is a secure authenticated key exchange with (weak) forward security under the assumption that the CDH problem is hard in the underlying group and the signature scheme being used is secure against chosen-message attack. The theorem provides the concrete relations between the advantages.

7.2 Sketch of proof of Theorem 1

Let $\mathbf{Z}$ be an adversary attacking the key distribution protocol in our model. We associate to it two other adversaries.

DH adversary. We describe a DH-adversary D who attempts to solve the computational Diffie-Hellman problem over $\mathcal{G}$. As described above, D gets inputs $\overline{X} = \overline{x}G$ and $\overline{Y} = \overline{y}G$, and is attempting to output a list of group elements that contains the DH-key $\overline{S} = \overline{xy}G$. In order to do this it will use $\mathbf{Z}$ as a subroutine. Our adversary D begins by picking random integers $g_{\rm l}, g_{\rm u}, l$ satisfying $1 \leq g_{\rm l} < g_{\rm u} \leq q_{\rm se}$ and $1 \leq l \leq q_{\rm H}$, where $q_{\rm se}$ is the number of Send queries made by $\mathbf{Z}$ and $q_{\rm H}$ is the number of H-queries made by $\mathbf{Z}$. It then starts running $\mathbf{Z}$. It will faithfully execute the experiment of running the adversary, itself choosing all static or ephemeral secret and public keys, with the following exceptions. Say the $g_{\rm l}$-th Send query is made to $\Pi^i_{\mathbf{U}}$ and the $g_{\rm u}$-th Send query is made to $\Pi^j_{\mathbf{V}}$. If either $\mathbf{U}$ or $\mathbf{V}$ is corrupted at the time of the Send query, D fails. Else it responds to the $g_{\rm l}$-th Send query by returning $\overline{X}$ as the ephemeral key, and to the $g_{\rm u}$-th Send query by returning $\overline{Y}$ as the ephemeral key. (The other parts of the flow, including the signature, are computed by D depending on the identity of the entity to which the query is sent.) If RevealSn queries are ever made to $\Pi^i_{\mathbf{U}}$ or $\Pi^j_{\mathbf{V}}$ then it fails. If not, however, it can complete the simulation. In that

case, it looks at the list of all H-oracle queries made. Each element of this list has the form $\mathbf{A} \parallel \mathbf{B} \parallel S$ where $\mathbf{A}, \mathbf{B}$ are identities and S is a group element. It edits the list to delete the identities, leaving a list of all group elements occurring in H-oracle queries, and outputs this list.

FORGER. We describe a forger F who has input pk and oracle access to $\mathrm{Sign}_{sk}(\cdot)$, and is trying to output a forgery relative to pk. It begins by picking at random an integer l in the range $1 \le l \le q_{\mathrm{en}}$ where q_{en} is the number of different entities that are active in the execution of $\mathbf{Z}$, as described above. It then begins running $\mathbf{Z}$, and sets the public key of the l-th entity invoked to pk. In the simulation, it chooses all quantities itself, and invokes its signing oracle whenever it needs to produce signatures under pk. If the entity whose public key is pk receives a $\mathsf{Corrupt}$ query then F fails. If some oracle receives a valid signed flow purporting to come from the entity whose public key is pk, but the corresponding message has not been queried of the signing oracle and the entity was uncorrupted at the time, then F outputs the message-signature pair as its forgery, and halts. (Note this entity may be corrupted later, and F would not be able to return the secret key sk, but F has succeeded and halted prior to this.)

ANALYSIS. We claim that

$$\mathsf{Adv}^{\text{ake-wfs}}_{P,LL}(\mathbf{Z}) \quad \le \quad q_{\mathrm{se}}^2 \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G}}(D) + 2q_{\mathrm{en}} \cdot \mathsf{Adv}^{\mathrm{sig}}_{SIG}(F)\ . \tag{1}$$

Taking into account the resource usage of the adversaries yields the theorem. We now very briefly justify Equation (1). In the "real" experiment of executing the adversary $\mathbf{Z}$, we define the event NA to be true if there exist $\mathbf{U}, \mathbf{V}, i, W, \sigma$ such that

- $\Pi^i_{\mathbf{U}}$ accepts flow $\mathbf{V} \parallel W \parallel \sigma$ and $\mathbf{U}$ was uncorrupted at the time this happens, but
- there do not exist j, τ such that $\Pi^j_{\mathbf{V}}$ output a flow $\mathbf{V} \parallel W \parallel \tau$ at a time when $\mathbf{V}$ was uncorrupted.

In the same experiment, the adversary eventually makes a Test query. Denote it by $\mathsf{Test}\,(\mathbf{U},\ i)$. We assume that the oracle $\Pi^i_{\mathbf{U}}$ to which the query is made had terminated and had a fs-fresh session key prior to this query, since otherwise the adversary loses. Let $\mathbf{V}$ be the PID of $\Pi^i_{\mathbf{U}}$. Let $(\mathbf{A}, \mathbf{B}) = (\mathbf{U}, \mathbf{V})$ if $\mathbf{U}$ played a sender role, and $(\mathbf{A}, \mathbf{B}) = (\mathbf{V}, \mathbf{U})$ if $\mathbf{U}$ played a receiver role. Let $X^* \parallel Y^*$ be the SID of $\Pi^i_{\mathbf{U}}$, and let $S^* = x^*y^*G$ be the associated DH key, where $X^* = x^*G$ and $Y^* = y^*G$. We now define the event CQ to be true if H-oracle query $\mathbf{A} \parallel \mathbf{B} \parallel S^*$ was made by the adversary $\mathbf{Z}$. Let $\textsc{win}(\mathbf{Z})$ denote the event that $\mathbf{Z}$ wins, meaning guesses correctly the value of the challenge bit b chosen by the oracle to which it issues the test query. Simple conditioning tells us that

$$\begin{aligned}
&\Pr[\,\textsc{win}(\mathbf{Z})\,] \\
&\quad = \quad \Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \overline{\mathrm{CQ}}\,\right] \cdot \Pr[\,\overline{\mathrm{CQ}}\,] + \Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \mathrm{CQ}\,\right] \cdot \Pr[\,\mathrm{CQ}\,] \\
&\quad \le \quad \Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \overline{\mathrm{CQ}}\,\right] + \Pr[\,\mathrm{CQ}\,]
\end{aligned}$$

$$
\begin{aligned}
&= \Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \overline{\mathrm{CQ}}\,\right] + \Pr\left[\,\mathrm{CQ} \mid \overline{\mathrm{NA}}\,\right] \cdot \Pr[\,\overline{\mathrm{NA}}\,] + \Pr[\,\mathrm{CQ} \mid \mathrm{NA}\,] \cdot \Pr[\,\mathrm{NA}\,] \\
&\leq \Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \overline{\mathrm{CQ}}\,\right] + \Pr\left[\,\mathrm{CQ} \mid \overline{\mathrm{NA}}\,\right] + \Pr[\,\mathrm{NA}\,] \,.
\end{aligned}
$$

Equation (1) follows from the following claims:

$$
\begin{aligned}
\Pr\left[\,\textsc{win}(\mathbf{Z}) \mid \overline{\mathrm{CQ}}\,\right] &= \frac{1}{2} && (2)\\
\Pr\left[\,\mathrm{CQ} \mid \overline{\mathrm{NA}}\,\right] &\leq \frac{q_{\mathrm{se}}(q_{\mathrm{se}}-1)}{2} \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G}}(D) && (3)\\
\Pr[\,\mathrm{NA}\,] &\leq q_{\mathrm{en}} \cdot \mathsf{Adv}^{\mathrm{sig}}_{SIG}(F) && (4)
\end{aligned}
$$

We omit justification of these claims.

8 Exposition

The exposition on SEC 1 is, by and large, excellent. The main problem with respect to ECDHS is what has already been discussed: that the goals associated to each scenario are not clearly specified.

There are very few typos or ambiguities. We enumerate some that we noticed.

1. Throughout. The use of *should* in this document should probably 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" → "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 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. x_G and y_G are undefined in the algorithm of 3.1.1.2.1. Add a definition of $(x_G, y_G) = G$.

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

11. 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?)

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

13. p. 33. It is not pa3-key TDES in CBC mode (with a 0-IV) is *not* designed to provide semantic security against CPA or CCA. Even with a random IV, the scheme definitely does not provide security against CCA.

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

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

16. p. 46, 48. "security level elliptic curve domain parameters" → "elliptic curve domain parameters"

17. p. 64. "point on E *on* large prime order" (should be *of*)

References

[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`

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *First ACM Conference on Computer and Communications Security*. ACM, November 93. Full version available from *http://www.cs.ucdavis.edu/∼rogaway*

[DS81] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24, 1981, pp 533–536.

[GMR84] S. Goldwasser and S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* 17(2):281–308, April 1988.

[K98] B. Kaliski. An unknown key-share attack on the MQV key-agreement protocol. Manuscipt. Proceedings version appeared in RSA Conference 2000 Europe, Munich, April 2000. Work based on earlier communication to IEEE P1363a and ANSI X9F1 workg groups, June 1998.

[SEC1] Certicom Research. Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography. Version 1.0, September 20, 2000. Certicom Corp.