SKEME: A Versatile Secure Key Exchange Mechanism for Internet

Hugo Krawczyk
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
hugo@watson.ibm.com

Abstract

A secure and versatile key exchange protocol for key management over Internet is presented. SKEME *constitutes a compact protocol that supports a variety of realistic scenarios and security models over Internet. It provides clear tradeoffs between security and performance as required by the different scenarios without incurring in unnecessary system complexity. The protocol supports key exchange based on public key, key distribution centers, or manual installation, and provides for fast and secure key refreshment. In addition,* SKEME *selectively provides perfect forward secrecy, allows for replaceability and negotiation of the underlying cryptographic primitives, and addresses privacy issues as anonymity and repudiatability.*

1 Introduction

The need to secure the Internet is now clear to everyone. Consequently, more and more mechanisms to provide security at different layers and different applications are being developed. Common to most of these security mechanisms is the need for key management. One crucial component of key management is the exchange of secret keys between two parties. This is a basic enabler for the use of conventional cryptography (based on shared secret keys) on which most of the efficient security solutions rely. This paper presents the design of SKEME, a key exchange mechanism which provides the scalability and flexibility required by the growing Internet, and by the diversity of scenarios that are part of it. Accommodating this variety of scenarios and requirements, both in terms of performance and security, into a simple and compact protocol is the main challenge and accomplishment of SKEME.

The design of SKEME has been mainly motivated by the work carried through the IPSEC working group of the IETF (Internet Engineering Task Force) which is in charge of developing a standard and interoperable key management protocol for Internet. This protocol is intended to provide key management for the IP-layer security protocols developed by the same working group (see [1]), but also to provide a key management solution for other security applications over Internet. The currently proposed mechanism for key exchange in this working group is the Photuris protocol designed by Phil Karn [17] which is similar to the STS key exchange protocol of Diffie, van Oorschot, and Wiener [11]. Photuris is designed to exchange a key between two parties using public key and the Diffie-Hellman key exchange [10]; it also addresses issues like anonymity and denial-of-service attacks.

SKEME provides with the same basic functionality of Photuris, it accommodates other trust models (e.g., key distribution centers or manual installation), and provides for flexible tradeoffs between security and performance (e.g., it allows for selective execution of Diffie-Hellman, and for fast and frequent key refreshment). Mostly importantly, the added features do not require any significant increase in protocol complexity, or incur on any performance penalty. Because of the many aspects that are common to Photuris and SKEME, the later is not proposed to replace Photuris, but to be merged with it in order to broaden and strengthen the functionality of the protocol.

1.1 Protocol Overview and Design Rationale

The design of SKEME follows a set of requirements and goals that are presented in detail in Section 2. Here we briefly overview these issues, and the basic features of SKEME.

The central approach of SKEME is to provide with a scalable and flexible protocol capable of accommodating many realistic scenarios in Internet, as well as to provide with clear tradeoffs between security and performance. The basic model that needs to be addressed is the public key model which is the most scalable and has minimal trust requirements. The basic mode of SKEME provides, therefore, key exchange based on public keys of the parties and the strong security of the Diffie-Hellman mechanism. The latter has the great advantage of minimizing the negative effects of the eventual exposure of long-lived keying material (e.g., the exposure of the private keys of the parties or a long-lived master key). This property of the Diffie-Hellman protocol is referred to as *perfect forward secrecy* (PFS) [11, 14].

However, SKEME does not limit itself to the combination of public key and Diffie-Hellman. It addresses additional needs like, e.g., key exchange based on a previously shared key between the parties. This supports many important and realistic scenarios which include manual key installation and other forms of shared master keys. Furthermore, it accommodates key exchange in the Key Distribution Center model (also known as the Kerberos model [20]), where the parties share a key via a commonly-trusted center. By using this key to authenticate a Diffie-Hellman exchange, rather than using it directly as a session key, SKEME achieves a significant security improvement by reducing the trust required in the KDC.

SKEME also provides for more efficient key exchange mechanisms for the cases in which the perfect forward secrecy property of the Diffie-Hellman exchange can be relaxed, thus saving the significant computational cost associated with that mechanism. Such cases include applications where authenticity of information, rather than confidentiality, is at stake, or where the level of secrecy required is relatively low (see Section 2.3.1). Finally, a very important goal of SKEME is to provide with a mode to perform very fast and frequent key refreshments. This has the effect of shortening the lives of cryptographic keys, thus limiting the damage and potential of key exposure.

All of the above results in four modes of SKEME:

1. The basic mode which provides both public key based key exchange and PFS (Diffie-Hellman).
2. Key exchange based on public keys but without performing the Diffie-Hellman algorithm.
3. Key exchange based on a previously shared key and provision of PFS.
4. Fast re-key mechanism based on efficient symmetric key techniques only (e.g., MD5).

These four modes are folded into a single simple protocol, with a small set of messages, a well defined set of options, and a compact representation. All of which makes SKEME attractive from a system and implementation point of view.

A basic technical observation behind the design of SKEME is that in most common implementations (e.g., RSA), the performance cost of public key signatures is similar to that of public key encryption. Therefore, SKEME deviates from the approach of performing a Diffie-Hellman exchange authenticated via digital signatures (e.g., [11, 16, 17]). Instead, the protocol first goes through a *share phase* in which the parties share a key by mutually encrypting half-keys based on the public key of each other, and then uses this key to authenticate the Diffie-Hellman exchange (against man-in-the-middle attacks). The later authentication utilizes efficient operations only, e.g., MD5. In this way the same functionality of Diffie-Hellman-plus-signatures is obtained, but one gets "for free" (i.e., with no further computational cost) the share phase where a key is shared by the parties. This allows for the option of skipping the Diffie-Hellman phase (thus providing mode 2 above), or skipping the share phase (i.e., omitting the use of public keys) when the exchange is to be based in a previously shared key between the parties (mode 3). Finally by skipping both the share phase and the Diffie-Hellman phase, a fast re-key mechanism is obtained (mode 4). For the details, see the protocol description in Section 3.

In addition, SKEME provides for anonymity in the sense that it prevents unnecessary disclosure of the communicating identities (beyond what is strictly necessary, e.g., IP addresses); it addresses further privacy issues by avoiding the use of digital signatures and then allowing for "repudiation of communication" (see Section 2.3.2); and it also provides certain defenses against denial of service attacks (by adopting the "cookies" technique of [17]).

Related work: Besides the works cited above, the literature contains many references to protocols and standards on key management in general and key exchange mechanisms in particular. We refer the reader to the survey by Rueppel and van Oorschot [21] on key exchange mechanisms, as well as to the forthcoming book by Menezes, van Oorschot, and Vanstone (chapter 12) [18]. Our work was mainly motivated by the activities carried through the IPSEC working group in the IETF, but we also note the suitability of our approach to the framework defined by IEEE 802.10 [15]. SKEME evolved as an extension of MKMP (modular key management protocol) [9] where a modular approach to key management is suggested and a specific module for key refreshment based on a shared key between the parties is presented. In general, SKEME builds as much as possible, even for its public key based modes, on the existing solid design and analysis work developed for key exchange over the shared-key model (cf. [19, 8, 4]).

Security analysis: A detailed and formal security analysis of SKEME is beyond the scope of this paper which is intended to describe the protocol design and its rationale. However, for completeness, we include an informal outline of a proof of security for all four modes of the protocol. We base this outline on previous formal work developed for proving the security of key exchange mechanisms in the shared-key model [4]. See Section 5. (We believe that the fact that the security of SKEME can be related to that of protocols in the shred-key model is another advantage of SKEME. The shared-key model has been extensively studied and we consider it simpler and better understood for analysis than the general public key case.)

Organization of this paper: Section 2 describes the main goals and requirements standing behind the design of SKEME. Section 3 describes the protocol. Section 4 highlights the main features of SKEME. Section 5 argues about the security of the protocol. Finally, Section 6 provides some concluding remarks, and comparison to Photuris.

A note on implementation: This paper presents the basic design and rationale of the protocol. Implementation of the protocol is underway; when completed more details and implementation experience will be reported in a separate document.

2 Goals and Requirements

In this section we list some of the basic goals and requirements behind the design of SKEME. It is not intended as an exhaustive enumeration of requirements for key management, but rather to emphasize the basic properties required from a key exchange mechanism for use in Internet, and to highlight the main features of the protocol proposed here. Many of these requirements are the result of the numerous discussions carried in the IPSEC working group of the IETF.

The basic functionality of a key exchange mechanism is to provide two communicating parties with a common shared secret key that is known to these parties only. One can think of this key as a master key or session key depending on the life span of the key or the semantics of the application. For the sake of concreteness, we will often refer to the exchanged key as a "session key".

2.1 Variety of Scenarios and Security Models

One of the most demanding aspects of a security protocol intended for use over the global Internet is the variety and heterogeneity of scenarios and security models existing in this medium. Any single model of trust will be insufficient to address the needs of key management for Internet. Our protocol is therefore designed to support a variety of these models while keeping the system aspects of the protocol as simple as possible.

PUBLIC KEY MODEL. In the case of two parties that possess the public keys of each other, the protocol is required to establish a shared secret key between these parties even if no prior shared secret existed between them. The security of the protocol will then depend on the authenticity of these public keys. The trust on this authenticity will be based, in general, on the existence of some form of certification mechanism that can range from a global Certification Authority to local and distributed trust as in the PGP model [22] (manual distribution of public-keys is also possible). The advantage of the public key model is that it has minimal third party trust requirements and, fundamentally, is the most scalable model for Internet.

KEY DISTRIBUTION CENTERS. A well established and extensively used trust model is that of a key distribution center (KDC) [19]. This is the model popularized by the Kerberos protocol [20]. Although it does not have the scalability capability of the public key model (because of the high level of trust required by the communicating parties on the on-line KDC), this model cannot be ignored in a key management protocol for Internet due to its usefulness in many scenarios (e.g, corporate world), the efficiency of symmetric cryptography techniques on which it is based, and its existing and extensive deployment. Actually, a well designed key exchange protocol can strongly improve the security provided by the KDC model by reducing the level of trust required from the KDC. This issue is further discussed in Section 2.3.1.

MANUAL KEY INSTALLATION. Although it is the most primitive way to install a shared key between two parties, manual installation is still a common way to install initial keys in many systems. (It will be also useful in the new scenarios created by Internet, e.g., a mobile user which shares a manually installed key with its own office's workstation, or with a firewall.) In this case, the use of a key exchange protocol that is able to do periodic refreshment of keys and minimize the security risk attached to long-lived master keys is essential.

We note that the basic difference between the public key case and the last two cases is the fact that in the latter the key exchange protocol starts once a shared secret key is already established between the parties. Periodic key exchange based on previously shared keys is useful also in the public key case, since it allows for more efficient, and then more frequent key refreshment. The latter minimizes compromise of keys via cryptanalysis (the more a key is used the more information the adversary has for breaking it) and limits the harm of an exposed key by shortening its useful life. In addition, key refreshment can minimize the dependency between past and future keys (i.e., how much can be learned from a compromised key about past and future keys).

2.2 Basic Requirements

In addition to support the above different models we briefly list here some of the properties SKEME is required to support.

SECRECY AND AUTHENTICITY. The protocol needs to guarantee to the executing parties that only the intended party learns the key exchanged and that this key is fresh and unique. Secrecy and authenticity (of the exchanged key) need to be protected against passive (eavesdroppers) and active (man-in-the-middle) attackers, and these properties be guaranteed for as long as the underlying cryptographic functions in use (encryption, authentication, etc) are secure against these adversaries. Not only the value of the exchanged key is to be protected, but partial information on the key should be hidden as well (except for known public information like key length)[1].

An additional security goal is to minimize the negative effects of a compromised key. Keys may be exposed regardless of the security of the key exchange protocol that generates them, e.g., by break-ins to a system, poor secure storage management, and so on. Mechanisms like independence between different keys in the system, frequent refreshment, and perfect forward secrecy, as discussed below, address this goal.

KEY REFRESHMENT. The key exchange protocol must provide automatic mechanisms to periodically refresh keys. This needs to include low cost mechanisms for very frequent key updates (say, each 5 minutes) and more costly and secure ones for less frequent refreshments (say, each 3 hours). These different mechanisms will differ in their performance cost and degree of independence between the refreshed key and past/future keys. Periodic refreshment of keys is required to limit the damage caused by exposed session keys and to reduce the amount of information

[1] Giving away even limited information about the key may have severe security implications. As an example consider the case where the exchanged key is later used as a one-time pad to hide one of two possible strings: the all zeros string or the all ones string (which may represent a confidential yes/no information); in this case, a single known bit in the key would reveal the encrypted message.

available for cryptanalysis (the latter is especially significant when weak algorithms are in use, e.g. due to performance or crypto regulations restrictions).

PERFECT FORWARD SECRECY. Perfect forward secrecy (denoted PFS) is a central notion pertaining key exchange protocols. It refers to the property that "disclosure of long-term secret keying material does not compromise the secrecy of exchanged keys from earlier runs." [11]. This property has a central role in our protocol and it is extensively discussed in Section 2.3.1.

KEY SEPARATION. Different cryptographic functions should use different and cryptographically independent keys (namely, the exposure of one key should not compromise the other). This applies to the different functions used in the key exchange protocol itself as well as to the cryptographic functions applied to data during the subsequent session. In particular, one has to be careful not to "re-use" the session key produced by the protocol to key different functions. (For example, if the session key is used in the protocol to key a hash function, say MD5, it should not be used to key a data encryption algorithm later, say DES.) The recommended way to use the session key to derive keys for different cryptographic algorithms is to use this session key as a "seed" from which other keys are derived (e.g., through a pseudorandom or hash function keyed with the session key and applied to a unique identifier of the algorithm).

PRIVACY AND ANONYMITY. Communication over a public and open network as the Internet not only requires the hiding of information exchanged between communicating parties but, in some cases, also the hiding of the identity of the communicating parties. This is not always possible: e.g., if a message carries for its delivery the IP address of the intended Internet host, or its originator, then that host can be identified from that information. However, in some other cases the communicating parties (typically, the initiator of a key exchange) can be hidden behind a temporary IP address, e.g., a traveling individual which is provided with a temporary address in a remote system. In such a case the sole information on the origin IP address is not sufficient to identify the communicating party. It is a goal of SKEME to protect this identity from attackers on the net. Other forms of privacy concerns exist, e.g., the protocol should *not* provide "non-repudiatable proofs" that party A has been talking to party B. The later is a real concern if digital signatures are used for authentication in the protocol. See further discussion of this issue in Section 2.3.2.

CLOGGING ATTACKS. The complete elimination of denial of service attacks via flooding or clogging against a host is virtually impossible. However, some preventive measures can alleviate the problem. In the case of protocols involving public key operations, the opportunity for such attacks is increased due to the high performance cost of these operations. In our protocol we adopt a simple "cookies" technique introduced by Phil Karn in his Photuris protocol [17] in order to make it more difficult for an adversary to accomplish such an attack.

PERFORMANCE. Key exchange operations may strongly vary on their computation requirements depending on the trust model and level of security required (compare an MD5 operation vs. an RSA signature). The protocol requires a careful design to provide flexible tradeoffs between security and performance. The different variants for key refreshment as mentioned above are an example of these tradeoffs. The amount of communication in the protocol is another performance parameter to consider.

MULTIPLE SECURITY MODELS. The protocol should support a variety of existing and widely used security models, like manual key installation and key distribution centers. In addition it must support public key based key distribution which is the only model to scale to the dimensions of Internet. The latter must be supported all the way from manually installed certificates to web of trust a la PGP to global certification authorities. These issues are discussed in more detail in Section 2.1.

ALGORITHM INDEPENDENCE. The protocol needs to define the underlying cryptographic primitives in a functional level (e.g., encryption, signature, etc.), but not to depend on particular implementations (e.g., DES, RSA). Specific realizations of cryptographic primitives should be replaceable. This would accommodate different choices by different parties to implement these primitives, and mostly importantly, would allow for replacement of the algorithms in case of cryptanalysis or the finding of new and more secure or efficient algorithms. This is not to preclude the definition of default algorithms that may be required for interoperability. Related to this issue is the need to include in a complete key management protocol a negotiation mechanism through which parties agree on particular security transforms and options of a protocol.

EXPORTABILITY. The reality of existence of regulations concerning import and export of cryptographic technology in different countries needs to be taken into consideration. The goal is not to weaken the protocol in order to make it exportable (as in the case of US regulations) but to design it such that it does not contain elements that will have to be unreasonably weakened because of these regulations. One such example is the avoidance, if possible, of mandatory symmetric encryption in the protocol.

MINIMIZE PROTOCOL COMPLEXITY. The above set of requirements, especially the need to support different security models and different tradeoffs security/performance may easily lead to a too complex protocol with too many options, message formats, and so on. In order to be acceptable and widely deployed the protocol needs to keep a low system complexity. This is the main challenge and accomplishment of SKEME: It accommodates the stated requirements through a small set of well-defined options, a uniform message format for the different options, and a compact representation.

2.3 Discussion of Some Requirements

While many of the above requirements are clear and do not necessitate much of an explanation, others are less well-understood and sometimes controversial. We elaborate on some of them here.

2.3.1 Perfect Forward Secrecy (PFS)

An important goal in a security design (especially key management) is to limit the harm caused by the exposure of keys. This is especially important for long-lived keys. If the compromise of a single key exposes to the attacker all the traffic exchanged by a party during, say, the last two years, such a key becomes an attractive target for an adversary, and a major bottleneck for the system security. A far better design is to limit the advantage for the attacker that breaks the key only to *future active* impersonation attacks, where the potential of being detected is high. A key exchange mechanism that protects short-lived keys from compromise even in case of the exposure of long-lived keys, is said to provide *perfect forward secrecy* (PFS) [11, 14].

If, for example, all session keys exchanged by a party, A, are encrypted under A's public key, then an attacker that breaks the private key of A would also learn all past, and even future, session keys of A. In contrast, by using the Diffie-Hellman algorithm for key exchange and A's private key only to sign this exchange, a much better level of security is achieved. In that case, the attacker that compromises the private key will be able to actively impersonate A in future communications, but will learn nothing about past communications, or even future ones in which the attacker is not actively involved.

Therefore, PFS is, in general, a very desirable property of a key exchange protocol. However, there is a computational cost to achieve it. Practically speaking, there is currently no other solution to provide this property except for the Diffie-Hellman exchange (in its different realizations: finite fields, elliptic curves, etc.)[2] which requires two long (modular) exponentiations for each party per exchange (one of which can be done off-line).

Due to this cost it is worthwhile asking if PFS is necessary in all cases and all scenarios. Our answer is no. PFS relates to *secrecy* of information. In some applications, however, authenticity rather than secrecy is the goal. As an example, the Authentication Header standard developed by the IETF [2] is intended to provide for authentication of IP packets and headers but not for confidentiality. A server that provides with non-confidential but authenticated information can use this authentication mechanism, but can dispense of secrecy, let alone PFS. In such a case, it could be unjustified to overload this server with the unnecessary effort of performing Diffie-Hellman in *each* key exchange (which may be required for each information request). Services requiring authentication but not confidentiality will be very common, e.g., a file server that authenticates files (especially, executables) for integrity, a web server providing non-confidential information, and so on. Other scenarios where PFS may not be a requirement include cases where the exchanged information is encrypted but its confidentiality is limited to a very short period of time (e.g., timely financial information), cases where the encryption in use is weak (for exportability, performance, etc.), or where the level of secrecy required is low (e.g., a DNS server that encrypts its responses for the sake of anonymity only).

SKEME is designed to *selectively* provide with PFS. It provides PFS as part of the basic SKEME protocol based on public key, and for cases where the parties perform key exchange based on long-term shared keys (like a manually installed master key, a SKIP key derived from long-lived public keys [3], and more). Furthermore, SKEME can provide PFS for the case of parties that share a common key via a key distribution center (KDC). In this case, SKEME would derive a session key for the parties via a Diffie-Hellman exchange, while the KDC-provided key would be used for authentication only. Thus, the compromise of the key provided by the KDC (e.g., by breaking into the KDC, or by malicious insiders) does not effect the security of information exchanged between the parties using the Diffie-Hellman key (it may only allow the attacker to mount limited active impersonation attacks). In contrast, if the KDC-provided key is used directly as a session key, its compromise would expose to the attacker all the traffic in that session. Therefore, providing PFS in this case strongly enhances security by reducing the trust required by the parties on the KDC.

On the other hand, for those cases, as discussed above, where the expense of a Diffie-Hellman exchange is not justified, SKEME provides with the option of omitting this mechanism and still providing the parties with a secret shared key. (To learn this key from the protocol the attacker will need to compromise the private keys of *both* parties). We consider the provision of selective PFS as one of the important differentiating properties of SKEME relative to other proposed key exchange mechanisms, e.g., [11, 17].

2.3.2 Privacy and the use of signatures

A natural approach to the design of a key exchange protocol based on public keys is to use digital signatures for authentication of the key exchange. Many protocols do that, e.g., [11, 16, 17]. Here we point out to some conflict between use of signatures and privacy requirements. A basic property of digital signatures is that they provide "non-repudiation"[3], namely, if A has signed some information using her private key, then anyone in possession of A's public key can verify that A was the signer. This makes digital signatures very useful in many scenarios, but also raises the fol-

[2] In principle, any public key system can be used to achieve such an exchange, including RSA. However, the latter would require the generation of one-time RSA keys which is an unacceptably expensive process.

[3] The term non-repudiation here does not necessarily imply "legal liability", but just the ability to give to a third party a strong evidence that the signed data originated with the signer.

lowing privacy concern. If during a key exchange between parties A and B, A is required to sign B's identity, then this signature can be used later by B (or any other party having that signature) to convince a third party that A had been communicating with B. The provision of such "proofs of communication" raise a privacy concern even more serious than disclosing identities over the network. Indeed, even if the identities are eavesdropped over the network, the eavesdropper cannot later provide convincing evidence to a third party that she saw these identities. She could provide that evidence, however, if these identities were signed by the communicating parties. (Even if the signatures are encrypted over the network, they can be used by the recipient B to show that A has been communicating with him).

Therefore, if digital signatures are used in a key exchange protocol (where privacy protection is a requirement), then these signatures should not be used to sign the identities. However, this will not prevent a malicious party from encoding her own name into a piece of information to be signed by the other party during the protocol. This encoding can be done through seemingly random nonces, or, in the case of a protocol signing Diffie-Hellman exponents, the encoding can be done via these exponents. In addition, the use of signatures imposes the need to encrypt these signatures before transmission to prevent an adversary from learning the identity of the sender just by checking that the signature is consistent with the public key of that party.

A better way to deal with these issues is to completely avoid the use of digital signatures in the protocol. The latter is the approach of SKEME. In our protocol, public key operations are limited to encryption/decryption, while all authentication of information by the parties is done through symmetric key techniques which provide complete repudiatability of the authenticated information, as well as avoids the need to encrypt the authentication. (For alternative public key-based exchange protocols that do not use signatures see [21].)

3 The SKEME Protocol

In this section we present the SKEME protocol with its basic phases and messages. The description is at a high-level and omits some of the details in order to concentrate on the basic security structure of the protocol and the functionality it provides. We first (§3.1) introduce some notation and the basic cryptographic functions underlying the protocol. Then (§3.2) we present the basic protocol which represents its strongest mode and provides with full security strength. Next (§3.3), the additional modes of SKEME are described in which the basic protocol is accommodated to provide flexible security/performance tradeoffs, and to support diverse security scenarios. Finally (§3.4), the full protocol is presented where the above modes are all combined into a single and compact representation.

3.1 Cryptographic Primitives and Notation

Here we list the basic cryptographic functions used in SKEME. We assume familiarity of the reader with these basic functions and concepts. Public key encryption is used, but no specific algorithm is specified or assumed. RSA or El Gamal encryption are examples of suitable algorithms. RSA is the most economic in the sense that only the operation of decryption involves a long modular exponentiation; encryption can be done with a few modular multiplications. The assumption on the security of the encryption is that it hides all partial information on the encrypted data (formally, we will assume *semantic security* of the encryption function [13]). In particular, it is assumed that encryption is randomized in the sense that the same message encrypted twice would lead to different encryptions. When using RSA this may be achieved by padding information with random salt before encryption, or more securely, by using the encoding scheme proposed in [6]. We use $PKE_A(\mathsf{info})$ to denote the (public key) encryption of info under the public key of party A.

SKEME uses the Diffie-Hellman key exchange algorithm[4] [10]. For simplicity of notation, we will denote the Diffie-Hellman exponents by $g^x \bmod p$ and $g^y \bmod p$ which corresponds to the standard Diffie-Hellman modulo a prime number p and generator g. However, there is no assumption in the protocol related to this special form. In general, one can use different primes/generators for different users, or use Diffie-Hellman over other structures (e.g., elliptic curves). Also for simplicity, we sometimes omit the mod p notation.

SKEME requires the use of *pseudorandom functions*, which are collections of keyed functions (like DES or keyed-MD5) with the property that their output (including individual bits of it) cannot be predicted by an adversary that does not possess the key to the function. These functions (formalized in [12]) extend the notion of pseudorandom generators, and can be seen as providing "random access" to a long pseudorandom string. In practice, pseudorandom functions have realizations via DES and other block cipher cryptosystems (in particular, can be applied to variable length input via CBC-MAC modes), or via *keyed* one-way hash functions like keyed-MD5. Two important properties of pseudorandom functions is that revealing the result of such a function on a set of inputs does not reveal information about the value of the same function on a different point, and that they serve as secure MAC (message authentication codes). These functions are used in SKEME both to provide data authentication (integrity) and as generators of keys. We denote a pseudorandom function using key K by F_K. In practical terms, one can think of F_K as keyed-MD5, DES-CBC-MAC, etc, and then very efficient to compute.

Finally, we need some further notation: H stands

[4] Any other exchange algorithm that provides perfect forward secrecy can be used instead of Diffie-Hellman; however, no such practical algorithm seems to be known — see 2.3.1.

for a strong one-way hash function, e.g., MD5, SHA. The parties to the protocol are referred to as A and B, where A acts as initiator, and their identities in the protocol are denoted by id_A, id_B, respectively. The output of the protocol, namely, the exchanged secret (session key) key, is denoted SK.

3.2 The Basic Protocol and Its Phases

There are three basic phases in the protocol: SHARE, EXCH, and AUTH. We first present these basic phases in a way that provides the full security capability of the protocol. This includes the sharing of a secret key based only on public keys and the provision of perfect forward secrecy. We call this the *basic protocol*. Later we show how these same phases with different parameters provide for the additional functionality and tradeoffs of the protocol. In the actual protocol messages of different phases can be combined to provide a more compact and communication efficient scheme. Some details are omitted for the sake of clarity of presentation.

Phase SHARE is intended to establish a key K_0 between A and B based only on the parties having the public key of each other. This phase by itself does not authenticate the parties or the shared key K_0. It provides a very basic security level: If A follows the protocol then she is assured that the shared key K_0 is not known to anybody except B (though A does not have the assurance that B knows the key). And analogously for B. To be really meaningful this phase needs to be combined with the other phases of the protocol.

In SHARE the parties exchange "half-keys" encrypted under each other's public key and then combine the half-keys via a hash function to produce K_0. (Exclusive-or of the two half-keys can be used instead of hashing, but the hashing provides less opportunity for B to arbitrarily influence the key). The basic message structure of SHARE is as follows:

SHARE:

$$\begin{aligned} &A \to B:\ \ PKE_B(K_A) \\ &B \to A:\ \ PKE_A(K_B) \\ &K_0 = H(K_A, K_B) \end{aligned}$$

When anonymity of the initiator, A, is desired then the first flow will also encrypt A's identity.

$$A \to B:\ \ PKE_B(id_A, K_A)$$

If id_A is too long, in particular if it includes A's public key certificate, then a second key can be included under the encryption, and id_A transmitted encrypted under this second key using a symmetric encryption algorithm. (Notice that this would require use of symmetric encryption in the protocol. However this is done only optionally and with no other security function than the hiding of identity/certificate of A). We stress that the values K_A and K_B need to be chosen as (pseudo-) random values, and fresh for each run of the protocol. [5]

The next phase, EXCH, is used to exchange Diffie-Hellman exponents. Notice that this phase is independent of SHARE.

EXCH:

$$\begin{aligned} &A \to B:\ \ g^x \bmod p \\ &B \to A:\ \ g^y \bmod p \end{aligned}$$

The above exponents g^x, g^y can be computed off-line by each party prior to the execution of the protocol.

The authentication of this Diffie-Hellman exchange is accomplished in the following phase, AUTH, which uses the shared key K_0 from SHARE to authenticate the Diffie-Hellman exponents. The combination of the EXCH and AUTH phases provides the protocol with the strong perfect forward secrecy (PFS) requirement.

AUTH:

$$\begin{aligned} &A \to B:\ \ F_{K_0}(g^y, g^x, id_A, id_B) \\ &B \to A:\ \ F_{K_0}(g^x, g^y, id_B, id_A) \end{aligned}$$

Remember (Section 3.1) that F_{K_0} represents a pseudorandom function using key K_0, and that such a function (e.g., keyed-MD5, CBC-MAC) provides the functionality of a MAC (i.e., message authentication). The messages of this phase are intended to authenticate the origin, the freshness, and the values of the Diffie-Hellman exponents. Only after this phase is completed, the parties have assurance that these values were chosen by their partner in the communication. Notice that the key K_0 shared in the SHARE phase can be known only to A and B (assuming their private keys are not compromised) and then only these parties could have generated the above authenticated messages. The inclusion of g^x in the first message serves to authenticate (to B) that g^x came from A; the value g^y in the same message is used to prove to B the freshness of this message (assuming g^y was freshly chosen by B); finally, the included identities serve to reassure the parties about the correct binding between the exchanged key and their identities. The second message has the same security functionality relative to A.

Computation of the session key: The session key SK, which is the key shared between A and B as the result of this protocol, is computed by the parties as $SK = H(g^{xy} \bmod p)$.[6] Notice that this computation (which involves an expensive on-line Diffie-

[5] An alternative to this public key-based SHARE phase is to use long term Diffie-Hellman public keys as described in [10, 3]. In this case, the public key of a party is of the form $g^s \bmod p$, and s is the secret key. The key K_0 used by two parties A and B is computed as $g^{s_A s_B} \bmod p$, where s_A, s_B are the private keys of A and B, respectively. The main drawback of such a key K_0 is that it remains unchanged as long as the parties do not change their public keys. For details on the use of Diffie-Hellman public keys to establish a shared-key see [3].

[6] We recommend hashing the value $g^{xy} \bmod p$ for the purpose of "extracting randomness" from all the bits of g^{xy}. Other

Hellman computation) can be completed after the protocol (thus, avoiding the consequent delay during the protocol). The value itself of *SK* is not used in the key exchange protocol. This saves the delay in the protocol that could be caused by this computation. The authentication of the individual exponents g^x and g^y guarantees the authenticity and uniqueness of *SK*.

3.2.1 Combining the Phases

The messages in the above phases have some level of flexibility as for their ordering and combination. For example, the ordering of messages in phase EXCH can be inverted, as is the case for messages in AUTH. In phase SHARE, A is assumed to be the initiator and then her message comes before B's. In some cases (discussed later) the order of phases SHARE and EXCH can be reverted. For the sake of communication efficiency, the above phases can be combined into three message as described in Figure 1.

Notice that, as presented, the above protocol represents an opportunity for an adversary to mount denial-of-service attacks against B, since B needs to perform an expensive operation (public key decryption) even before he knows to whom he is speaking, and before A authenticates herself. Moreover, in order to force B to decrypt the adversary does not need to perform any expensive operation. This problem is addressed in the Photuris protocol [17] via a technique called *cookies*. The same technique can be used in our case, it requires the performance of a COOKIES phase before the SHARE phase. Its incorporation to our protocol is straightforward and the details are omitted here. (We refer to [17] for further motivation and description of this technique).

3.3 Modes of the Protocol

As said earlier, the distinguishing feature of SKEME is its capability to provide the different tradeoffs between performance and security as required by the different security scenarios in Internet. The basic protocol as described in Section 3.2 provides the most general solution in the sense that it accommodates the public key model (which is the most general and scalable model) and achieves perfect forward secrecy, thus providing the strongest level of secrecy even in case of key compromise.

In this section we show some natural variations, or modes, of the basic protocol that provide with the suitable level of security for scenarios where a secret key is already shared between the communicating parties, and those scenarios where the requirement of perfect forward secrecy can be relaxed and, therefore, its high computational cost be saved. (See Figures 2 and 3 for a combined and pictorial representation of these modes.)

methods are possible, including using directly some of the bits of g^{xy}. However, notice that some of these bits, e.g. least significant bit, are predictable given g^x and g^y. Another possible method to compute *SK* is as $SK = F_{K_0}(g^{xy} \bmod p)$.

3.3.1 SKEME Without PFS

In Section 2.3.1 different scenarios where perfect forward secrecy (PFS) may not be a requirement were discussed. The mode of SKEME presented in this section is intended to provide the key exchange functionality based on public keys of the parties, but without paying the high performance cost required to achieve PFS. This mode takes the advantage of the key K_0 shared during the SHARE phase to produce a shared session key.

This mode of SKEME is derived from the basic protocol of Section 3.2 by modifying the EXCH phase. In this mode, the EXCH phase will be simply used to exchange *nonces* between the parties. These are (unstructured) random numbers freshly generated by the parties and sent instead of the Diffie-Hellman exponents. These nonces are typically implemented as pseudorandom bit strings of length 64-128. The resultant phases EXCH and AUTH follow the known techniques from the shared-key model originated with [19], and further developed and analyzed in subsequent works (especially, [8, 4]).

EXCH:

$$A \to B:\ nonce_A$$
$$B \to A:\ nonce_B$$

The AUTH phase is modified accordingly to replace g^x and g^y, with $nonce_A$ and $nonce_B$, respectively, as the F_{K_0} arguments.

AUTH:

$$A \to B:\ F_{K_0}(nonce_B, nonce_A, id_A, id_B)$$
$$B \to A:\ F_{K_0}(nonce_A, nonce_B, id_B, id_A)$$

In this way the combination of EXCH and AUTH provides the parties with the assurance that the key K_0 they shared through the SHARE phase is known to both of them, and that the party they talked to is the correct one (because only that party had the private key required to decrypt the corresponding half-key K_A or K_B). The nonces $nonce_A$ and $nonce_B$ act as *challenges* to "prove" to each other the possession of K_0.

Computation of the session key: One could imagine that the output of the protocol, i.e., the session key *SK* to be shared between the parties would be K_0. However, since K_0 is used in the protocol as the key to the function F, it should not be output as the session key, otherwise one would be giving away some information on *SK* before it is even used[7]. Therefore, instead of outputting $SK = K_0$, we define *SK* to be $F_{K_0}(arg)$, where *arg* is the value sent in

[7] As an example, assume that the key is used later, in a different protocol, where one of the parties proves to the other the possession of the key *SK* by just showing a pair $(x, F_{SK}(x))$ for any value of x. Had the key *SK* never been used before then the above would constitute a (one-time) proof of possession of the key; however, doing that with $SK = K_0$ would be insecure since the key exchange protocol itself provides such a pair.

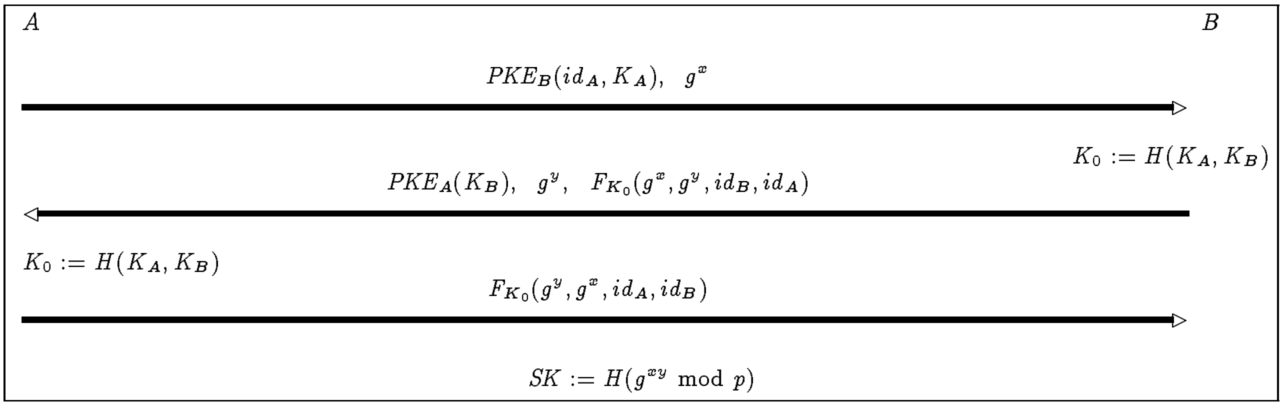

Figure 1: SKEME with public keys and PFS

the message from A to B in phase AUTH, namely, $arg = F_{K_0}(nonce_B, nonce_A, id_A, id_B)$. By the properties of pseudorandom functions the value of F_{K_0} on arg is (computationally) independent of any other value output by F_{K_0}. Thus, the protocol provides no information to an adversary on the value of SK.

Notice that in order to find SK from this mode of SKEME, an adversary needs to actively impersonate one of the parties (which may succeed only if the attacker possesses the private key of that party), or to be able to learn K_0 by watching the communication in the protocol. However, the latter requires knowing the private keys of *both* parties in order to decrypt K_A and K_B. Notice that for a passive attacker (eavesdropper only) knowing only one of the private keys is not sufficient. Therefore, although it does not provide PFS, this mode of the protocol provides a significant level of security without incurring the expense of a Diffie-Hellman exchange.

3.3.2 Pre-shared key and PFS

In this mode, the protocol assumes that the parties already share a secret key, and that they use this key in order to derive a new and fresh key (as discussed in Section 2, this previously shared key could come from a key distribution center, a manually installed key, a long-lived shared master key, etc.). This mode provides also with perfect forward secrecy (PFS), thus ensuring the parties that even a future compromise of their pre-shared key will not expose all the traffic encrypted with keys derived from the exchanged session keys.

In this mode of SKEME the SHARE phase can be skipped and the pre-shared key used as K_0. No further modification to the basic protocol of Section 3.2 is required.

Computation of the session key: The computation is identical to that of the basic protocol, namely, $SK = H(g^{xy} \bmod p)$.

3.3.3 Fast Re-Key

This is the fastest mode of SKEME. It is intended to provide for very frequent key refreshment without going through expensive operations like public key or Diffie-Hellman computations. It involves only fast operations like MD5. It assumes the parties share a key K_0 from a previous round of the protocol and they use that key to exchange a new one. The newly exchanged session key SK will have the property that its exposure does not expose any prior keys (however, an eventual exposure of K_0 would compromise all session keys exchanged under this mode due to its lack of PFS).

In this mode the SHARE phase is omitted, and the EXCH and AUTH phases are run as in Section 3.3.1, namely, by exchanging nonces rather than Diffie-Hellman exponents. (The resultant fast re-key protocol follows the authentication and key exchange techniques of [8, 4] based on symmetric key cryptography only.)

Computation of the session key: Computation of SK is identical to that of Section 3.3.1, namely, $SK = F_{K_0}(arg)$, where $arg = F_{K_0}(nonce_B, nonce_A, id_A, id_B)$ is the value sent in the message from A to B in phase AUTH.

3.4 The Combined Protocol

After having described the four modes of SKEME, we present here the complete protocol in which all these modes are combined into one uniform and compact format. This combined protocol is shown in Figure 2 and the derivation of the different modes of SKEME is summarized in the table in Figure 3. The inclusion in square brackets of the public key encryption pieces is done to stress that these elements are omitted in some of the modes of the protocol.

Figure 3 shows the following information: mode name, whether the public key encryption is performed or not (in particular, whether the mode assumes the existence of public keys of the parties or not), whether Diffie-Hellman is performed (thus providing or not PFS), and the mode-dependent values of $value_A, value_B$, K_0, and SK. Notice how the four modes of the protocol depend on the selection to perform or not the expensive operations required by the use of public keys (encryption) or to provide PFS (Diffie-Hellman).

Figure 2 illustrates how the different modes can be implemented through a uniform set of messages. Notice that the same message field can carry either a nonce or a Diffie-Hellman exponent, depending on the mode in use. These fields need to be, in any case, length-variable since the Diffie-Hellman exponents may have different lengths according to the required cryptographic strength. Also the fields carrying the public key encryption can be included or omitted according to the mode (if omitted they can be considered a zero-length field). Finally, the agreement on which mode of the protocol is to be used is part of the negotiation between the parties which also includes negotiation of specific security transforms.

Many implementation details are omitted here, including system issues (e.g., exact formats, acks, retransmission, security association identifiers, etc.) and cryptographic issues (e.g., off-line/on-line Diffie-Hellman computation, pseudorandom generation, format of public key encryption, derivation of multiple keys, etc.). Also, details of negotiation of modes, options and specific cryptographic primitives are omitted. All of these are beyond the scope of this paper which focuses on the basic design issues and rationale. However, we stress here that no security design is complete without a sound treatment of fault tolerance and error handling issues. We will further address these aspects when reporting on our implementation work. Many of the implementation details, however, are common or similar to the Photuris protocol [17] where some of them are already dealt with.

4 Summary of Main Features

SKEME is designed to achieve the requirements listed and discussed in Section 2. In particular, to provide support for the different security scenarios and to allow flexible tradeoffs between security and performance, while maintaining system complexity as simple as possible. In this section we briefly summarize those features whose combination differentiates SKEME from other proposed protocols (most notably, the STS/Photuris protocols [11, 17]). See Sections 2 and 3 for more details on these features.

Public key-based key exchange. Provides key exchange based on public keys, with and without PFS. In the first case, an eavesdropper does not learn the exchanged key even if it knows both private keys of the parties. In the latter, it requires the compromise of both private keys in order to learn the exchanged session key. Active impersonation of a party during the protocol is possible only by an adversary that possesses the private key of the party.

Support manual installation and other long-lived master keys. Supports key exchange based on a previously shared key between the parties. It provides PFS to this exchange in order to protect against information disclosure in case of exposure of a long-lived key.

Key exchange based on KDC model. Keys can be exchanged based on a common key distributed to the parties by a KDC. The security of the KDC model is enhanced via the provision of PFS to the key exchange (see Section 2.3.1).

Security. SKEME is designed to provide protection against passive and active adversaries, including man-in-the-middle and replay attacks (by using authentication and fresh randomness in each key exchange), and to limit the damage from exposed keys. Session keys do not leak information on other session keys (by virtue of the use of pseudorandom functions for the derivation of these keys), and even the effect of a compromised private key is limited. Such a compromise would allow the attacker to actively impersonate the compromised party, but will not allow that attacker to learn past keys. Indeed, when the parties perform a public key-based exchange without PFS, the adversary needs to know *both* parties' private keys to learn the keys. If the exchange includes PFS (i.e. Diffie-Hellman) then the only way for the attacker to learn a key is by actively impersonating the compromised party during the exchange. Other security features, like anonymity, privacy, and key separation are discussed in the sequel. See also Section 5 about the security analysis of SKEME.

Selective PFS. Perfect forward secrecy via Diffie-Hellman exchange provided as part of the basic protocol. It can be optionally omitted for the scenarios where its cost or functionality is not justified (see Section 2.3.1).

Fast re-key. A fast re-key mechanism based on symmetric cryptography only is provided and intended for very frequent refreshment of session keys in order to shorten key lives.

Performance. The basic mode of SKEME provides the best security functionality and is also the most expensive mode in the protocol. Using RSA for public key encryption it involves, for each party, one long exponentiation for decryption, and 2 exponentiations (one off-line) for Diffie-Hellman. In the pre-shared key modes, the public key decryption operations are saved, while in the share-only mode (without PFS) the two exponentiations per party of Diffie-Hellman are saved. The fastest mode is 'fast re-key' which involves symmetric key operation only (MD5-like).

We stress that the cost of a Diffie-Hellman exchange can be reduced by using short exponents. For mod p implementations we recommend a bare minimum of 160 bits for the exponents (the values x and y in our description), and preferably 256 bits[8]. Notice, also, that in SKEME the computation of g^{xy} can be performed (or completed) after the end of the protocol (since this value is not used by the protocol itself except for the computation of SK). This avoids the introduction of delays between protocol messages due to this long computation[9].

[8] As a preventive measure against partners to communication that use too short exponents (a fact that is hard to detect) SKEME could use a key derived from the SHARE phase to encrypt the Diffie-Hellman exchange (cf. [7]).

[9] We stress that in contrast to some other protocols (e.g.,

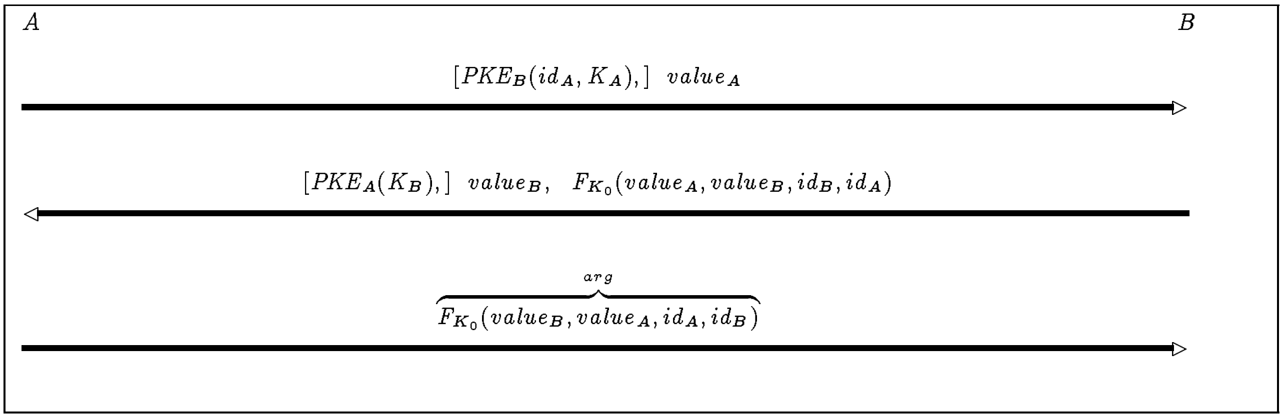

Figure 2: Combined SKEME Protocol

Mode	PK	DH	$value_A$	$value_B$	K_0	SK	Comments
basic protocol (full strength)	Y	Y	g^x	g^y	$H(K_A, K_B)$	$H(g^{xy})$	PK+PFS (see Fig. 1)
share only (no PFS)	Y	N	$nonce_A$	$nonce_B$	$H(K_A, K_B)$	$F_{K_0}(arg)$	Effic (no DH) No PFS
pre-shared key	N	Y	g^x	g^y	pre-shared (input to protocol)	$H(g^{xy})$	omit PK, e.g., Kerberos+PFS
fast re-key	N	N	$nonce_A$	$nonce_B$	pre-shared (input to protocol)	$F_{K_0}(arg)$	MD5-only no PK, no DH

Figure 3: SKEME Modes

Key separation. SKEME is carefully designed according to the key separation principle (see Section 2.2). SK is never used in the protocol. K_0 is used in the protocol only as a key to the function F. In the cases where K_0 is produced by the SHARE phase then we know that K_0 was not used before (and won't be used beyond the protocol). However, in the modes where K_0 is input into the protocol (the pre-shared mode and fast re-key) the implementation has to be careful not to use this key outside the protocol for anything else except keying the function F. (This would be satisfied if, e.g., K_0 is freshly distributed by a KDC, or if K_0 is a key solely used in consecutive runs of the fast re-key protocol.) As for the key SK, we recommend not using it to directly key a cryptographic algorithm applied to data during the session, but instead to use SK to derive independent keys for the different algorithms used during the session. As an example if a key for DES-CBC is required, then the key to be used can be derived from F_{SK}(des-cbc-id) where des-cbc-id stands for a unique identifier for this algorithm. We note that there are ways to derive a variable number of key bits using pseudorandom or keyed hash functions. (Details are beyond the scope of the paper.)

[11, 17]) there is no security or functional reason in SKEME to prove possession of the session key by the parties during the protocol. This possession is (implicitly) guaranteed by a successful (i.e., errorless) completion of the protocol.

Anonymity. The initiator, A, of the protocol may hide her identity whenever the mandatory plain information (e.g., IP address) is insufficient to identify her. This feature is achieved in a 'natural' way by encrypting A's identity, in the first message, together with the half-key exchange. This property is not provided for key exchange based on previously shared keys; in these cases, sending the parties identities is unnecessary since the parties identify themselves via IP addresses and previously exchanged security association identifiers. Identification of the receiver is not disclosed by the protocol except for the necessarily transmitted plain information (IP address).

Signature-less property. The protocol does not use digital signatures. This avoids privacy issues as the need to encrypt these signatures for anonymity, and most importantly, avoids providing (forced) "proofs" of communication between the parties (see Section 2.3.2). Notice that authentication is carried in SKEME via MAC or pseudorandom functions, and then it provides no useful proof of communication to third parties.

Denial of service attacks. Potential clogging attacks are alleviated through the cookies technique of [17].

Use of symmetric encryption. There is no requirement for use of symmetric encryption inside the protocol. It may be optionally used to provide anonymity in the case in which A transmits her own public key certificate to B in the first message of the protocol, but has no other security implications.
Use of public key encryption. Our protocol uses public key encryption to allow for key exchange without recurring to Diffie-Hellman (costly) algorithm. It is also used as a replacement for signatures when running Diffie-Hellman; in this case the key (K_0) exchanged through the use of encryption is used *only* for the purpose of authentication. We note that public key encryption is regulated in some countries. However, it is generally allowed for use for the purpose of key distribution, and as such subject to similar restrictions as a Diffie-Hellman exchange. We note that security can be enhanced by periodically renewing the public key encryption keys; e.g., by self-certifying them using a long-term certified signature key.
Algorithm independence. The protocol requires several cryptographic primitives, but does not depend on any specific implementation of these primitives (only on the basic requirements from these functions as specified in section 3.1.
Protocol complexity. In spite of the broad functionality it provides, the protocol has a simple and compact form. Different modes are supported with a small and uniform set of messages.

5 Security Analysis

Although this paper concentrates on the description of the protocol and its requirements, and not on a detailed security analysis, we present here some brief arguments supporting the security of SKEME. (Our line of analysis is based on generic security requirements from the underlying cryptographic functions rather than on specific implementations of these functions. In this way we achieve true algorithm independence.)

We need to argue for the security of the four modes of SKEME. The fast re-key mode (section 3.3.3) follows the well-understood and analyzed protocols in [8, 4]. In particular, the paper by Bellare and Rogaway [4] (see AKEP2 protocol in that paper), contains a rigorous proof of security directly applicable to this mode of SKEME. It follows that this mode is secure (for secrecy and authenticity) as long as the underlying cryptographic primitives are secure in the sense defined in section 3.1.

We argue that the analysis in [4] can be adapted to prove the mode of SKEME (section 3.3.2) in which the parties share a key K_0 prior to the protocol and use this key to authenticate the Diffie-Hellman exchange. This protocol can be "mapped" to the protocols analyzed in [4] by considering the Diffie-Hellman exponents (g^x and g^y) as the nonces in the protocol. If instead of generating the session key SK by $H(g^{xy})$ we would do it by applying the pseudorandom function F_{K_0} to g^{xy}, then the extension of the analysis of [4] is straightforward. (A suggestion along these lines was first made in [5].) There is no problem to modify our description of the protocol to use F_{K_0} as above. On the other hand, the proof can be extended also to the case $SK = H(g^{xy})$, if we assume the Diffie-Hellman conjecture on the unpredictability of $H(g^{xy})$ given g^x and g^y (a conjecture required anyway to claim the perfect forward secrecy of the basic Diffie-Hellman exchange).

The other two modes of SKEME are related to the above. The difference is that instead of assuming a pre-shared secret key K_0 between the parties, it is SKEME itself that generates this key through the SHARE phase. That phase, by itself, does not guarantee anything except that if there are any parties that know the key $K_0 = H(K_A, K_B)$ after the execution of SHARE then those parties are A and/or B. (This is implied by the security of the encryption function *PKE*.) Therefore, the premise that K_0 is not known to anybody except for A and B, which is the basis for the proof in the case of a pre-shared key, holds here too. (We stress that the successful run of the AUTH phase serves by itself as a confirmation that A and B indeed learned the key K_0 through the SHARE phase.) This is clearly an informal argument. Formalizing these ideas requires extending the definitions of security from the shared-key model to public key. Based on such definitions, a formal proof would show that any adversary that can break the security of the protocol (by impersonation, learning information on the key, etc.) can be transformed into an adversary that is able to break (one or more of) the underlying cryptographic functions (e.g., the public key encryption, the pseudorandom function, etc.). Then, the security of SKEME would follow from the security of these cryptographic primitives.

Note: We have assumed as part of the above analysis that the public Diffie-Hellman exponents (g^x, g^y) exchanged between the parties can be seen as nonces. This assumes that these public exponents are unique per exchange. This is indeed very desirable since the perfect forward secrecy property of the Diffie-Hellman exchange calls for the destruction of the secret exponent, say x, immediately after the Diffie-Hellman key is computed. However, it has been suggested (e.g., [17]) that a party could reuse the same secret and public exponent for different exchanges during a relatively short period of time (say, few minutes). The reason for such reuse is to amortize the cost of computing the exponentiation $g^x \bmod p$ over several exchanges. We suggest the following strategy for an implementation that wants to achieve these savings. It will periodically generate $g^x \bmod p$ with a new x (say, each 5 minutes) and will use $g^x \bmod p$ as the public exponent for the first exchange in that period, $g^{x+1} \bmod p$ for the second, $g^{x+2} \bmod p$ for the third, and so on. Each new public exponent in this sequence costs a single modular multiplication but results in a fresh value (nonce) for the sake of authentication. Moreover, two parties that perform two successive exchanges in a short period of time are guaranteed in this way to compute different Diffie-Hellman keys, and then completely different session keys (which are computed as the hash value of these two different Diffie-Hellman keys).

6 Concluding Remarks

We have presented a secure and versatile protocol for key exchange suitable to support a wide variety of scenarios, security models, and security-performance tradeoffs, as well as a detailed discussion of security requirements for such a protocol. The main motivation for the present work has been the ongoing effort in the Internet community to standarize key management mechanisms to support secure IP (Internet Protocol). However, this work has applicability to many other environments as well, e.g., the framework of IEEE security protocols [15].

The protocol has many similarities, and some important differences, with Photuris [17], which is being developed through the IPSEC working group of the IETF for the above purpose. Both protocols provide as their basis an authenticated Diffie-Hellman exchange based on public key. Photuris does it by first performing a Diffie-Hellman exchange and then authenticating it through the use of digital signatures (similarly to the STS protocol of [11]). SKEME uses public key encryption to exchange a one-time key and then uses shared-key techniques to authenticate the Diffie-Hellman exchange. An important advantage of the SKEME approach is that it allows for selective performance of the (expensive) Diffie-Hellman operations. That is, in SKEME one can skip the Diffie-Hellman phase and still have a key exchanged between the parties; in Photuris that is not possible.

Photuris was originally designed with less functionality in mind than SKEME. One of the main motivations in developing SKEME (whose basic ideas were first presented by this author through the IPSEC working group) was to promote the addition to the Photuris protocol of some of the elements presented here. These elements include the support of shared-key models (mainly, manual installation and key distribution centers) and the performance of cheap and frequent re-key operations (based on fast symmetric key techniques only). Some support for these aspects has since then been added to Photuris. The latter would be further benefited by the adoption of the specific mechanisms provided in SKEME for the pre-shared key modes, including fast re-key (mechanisms which follow well-known and analyzed techniques from previous works). In addition, the support in Photuris for selective Diffie-Hellman performance is strongly recommended.

Other advantages of SKEME over Photuris include the provision of an anonymity mechanism which does not require the use of symetric encryption and is secure against active attackers (Photuris uses symetric encryption to hide signatures and cerificates, and relies on a non-authenticated Diffie-Hellman to derive the key for this encryption), dispensing of digital signatures (thus resolving the privacy issues raised by the use of signatures), and freeing the protocol from delays introduced by the costly computation of the Diffie-Hellman key (SKEME can complete that computation after the protocol execution, Photuris cannot since it uses that key during the protocol itself). In addition, we believe that SKEME is better suited for security analysis using current analysis techniques, especially since the analysis of SKEME can be strongly related to the analysis work already developed for symetric key protocols (see Section 5).

Finally, we stress that SKEME can be readily used in conjunction with the SKIP protocol [3] (which has also been proposed in the context of the IPSEC working group). The latter uses long-term Diffie-Hellman public keys to derive long-term shared keys between parties. These shared keys can be used in SKEME as the key K_0 (which can be cached or re-computed upon use out of the Diffie-Hellman public keys). This would provide SKIP with much of the security functionality that it lacks now, especially perfect forward secrecy and frequent interactive key refreshment.

Acknowledgment

I thank Pau-Chen Cheng, Juan Garay, and Amir Herzberg, for many fruitful discussions on IP security and key management. The work presented in this paper is a follow-up to our joint work on the MKMP protocol [9]. I am particularly indebted to Pau-Chen for his ongoing implementation of SKEME. I also thank Shai Halevi, Paul van Oorschot, and David Wagner, for their excellent comments on earlier versions of this paper (though, of course, I carry full responsibility for all the paper's faults). Finally, I thank the program committee of ISOC SNDSS'96 for the thorough review of the paper.

References

[1] R. Atkinson, "Security Architecture for the Internet Protocol", RFC1825, August 1995.

[2] R. Atkinson, "IP Authentication Header", RFC1826, August 1995.

[3] A. Aziz, "Simple Key-Management for Internet Protocols (SKIP)," Internet Draft draft-ietf-ipsec-aziz-skip-02.txt, September 1995, work in progress.

[4] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," *Advances in Cryptology, - CRYPTO'93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249. (Full version available from the authors or from `http://www-cse.ucsd.edu/users/mihir`.)

[5] M. Bellare, and P. Rogaway, "Distributing keys with perfect forward secrecy", manuscript, Jan. 1994.

[6] M. Bellare, and P. Rogaway, "Optimal Asymmetric Encryption – How to encrypt with RSA", *Advances in Cryptology - EUROCRYPT'94 Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.

[7] M. Bellovin, and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks", *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, May 1992, pp. 72-84.

[8] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. "Systematic design of a family of attack-resistant authentication protocols," *IEEE Journal on Selected Areas in Communications* (special issue on Secure Communications), 11(5):679–693, June 1993.

[9] P. Cheng, J. Garay, A. Herzberg, and H. Krawczyk, "Design and implementation of modular key management protocol and IP Secure Tunnel on AIX", In *Proc. 5th USENIX UNIX Security Syposium*, Salt Lake City, Also available from ftp site software.watson.ibm.com: `/pub/security/mkmp-ipst-usenix.ps`.

[10] W. Diffie, and M. Hellman, "New Directions in Cryptography", *IEEE Trans. Info. Theory* IT-22, No. 6, pp. 644-654.

[11] W. Diffie, P. C. van Oorshot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges", *Designs, Codes and Cryptography*, V. 2, Kluwer Academic Publishers, 1992, pp. 107-125.

[12] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM,* Vol. 33, No. 4, 210–217, (1986).

[13] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.

[14] C.G. Günther, "An identity-based key-exchange protocol", *Advances in Cryptology - EUROCRYPT'89*, Lecture Notes in Computer Science Vol. 434, Springer-Verlag, 1990, pp. 29-37.

[15] IEEE 802.10c, "Interoperable LAN/MAN Security — Clause 3: Key Management Protocol", draft, March 1995.

[16] ISO/IEC IS 9798-3, "Entity authentication mechanisms — Part 3: Entity authentication using asymmetric techniques", 1993.

[17] P. Karn, and W. A. Simpson, "The Photuris Session Key Management Protocol", Internet Draft draft-ietf-ipsec-photuris-03.txt, September 1995.

[18] A.J. Menezes, P.C. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press (Boca Raton, Florida), to appear 1996.

[19] R.M. Needham, and M.D. Schroeder, "Using encryption for authentication in large networks of computers", *Comm. of the ACM*, 21, 1978, pp. 993-999.

[20] B. C. Neuman and T. Ts'o. "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications*, 32(9):33-38. Sept. 1994. `http://nii.isi.edu/publications/kerberos-neuman-tso.html`.

[21] R.A. Rueppel, and P.C. van Oorschot, "Modern key agreement techniques", *Computer Communications*, 17, 1994, pp. 458-465.

[22] P. Zimmermann, *PGP User's Guide.* Boulder, Colo. 1994.