Authenticated Diffie–Hellman Key Agreement Protocols

Simon Blake-Wilson[1] and Alfred Menezes[2]

[1] Certicom Research, 200 Matheson Blvd. W., Suite 103
Mississauga, Ontario, Canada L5R 3L7
sblakewi@certicom.com
[2] Department of Combinatorics & Optimization
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
ajmeneze@cacr.math.uwaterloo.ca

Abstract. This paper surveys recent work on the design and analysis of key agreement protocols that are based on the intractability of the Diffie–Hellman problem. The focus is on protocols that have been standardized, or are in the process of being standardized, by organizations such as ANSI, IEEE, ISO/IEC, and NIST. The practical and provable security aspects of these protocols are discussed.

1 Introduction

Authenticated key establishment protocols are designed to provide two or more specified entities communicating over an open network with a shared secret key which may subsequently be used to achieve some cryptographic goal such as confidentiality or data integrity. Secure authenticated key establishment protocols are important as effective replacements for traditional key establishment achieved using expensive and inefficient couriers.

Key establishment protocols come in various flavors. In *key transport* protocols, a key is created by one entity and securely transmitted to the second entity, while in *key agreement* protocols both entities contribute information which is used to derive the shared secret key. In *symmetric* protocols the two entities a priori possess common secret information, while in *asymmetric* protocols the two entities share only public information that has been authenticated. This paper is concerned with two-party authenticated key agreement protocols in the asymmetric setting.

The design of asymmetric authenticated key agreement protocols has a checkered history. Over the years, numerous protocols have been proposed to meet a variety of desirable security and performance requirements. Many of these protocols were subsequently found to be flawed, and then either were modified to resist the new attacks, or were totally abandoned. After a series of attacks and modifications, only those surviving protocols which had received substantial public scrutiny and were believed to resist all known attacks were deemed secure for practical usage. Protocols that evolve from this 'attack-response' methodology are said to provide *heuristic* security.

S. Tavares and H. Meijer (Eds.): SAC'98, LNCS 1556, pp. 339–361, 1999.

There are two primary drawbacks of protocols which provide heuristic security. First, their security attributes are typically unclear or not completely specified. Second, they offer no assurances that new attacks will not be discovered in the future. These drawbacks make a notion of *provable* security desirable. This would entail specification of a formal model of computing which accurately captures the characteristics of the participating entities and a real-life powerful adversary, a formal definition of the security goals within this model, a clear statement of any assumptions made, and, finally, a rigorous proof that the protocol meets these goals within the model.

While provable security may appear to be the highest possible level of security for a key agreement protocol, the approach does have some limitations. Most significantly, it is difficult to judge whether or not real-life threats and security goals are adequately reflected in a given model. That is, the provable security of a protocol is meaningful only if one finds the model, definitions, and underlying assumptions to be appropriate for one's purposes. Nevertheless, significant progress has been made in recent years, and authenticated key agreement protocols which are both provably secure and efficient have been devised and are being used in practice.

This paper focuses on asymmetric authenticated key agreement protocols whose security is based on the intractability of the Diffie–Hellman problem. We discuss the practical and provable security aspects of some protocols which are being standardized by accredited standards organizations such as ANSI (American National Standards Institute), IEEE (Institute of Electrical and Electronics Engineers), ISO/IEC (International Standards Organization/International Electrotechnical Commission), and the U.S. government's NIST (National Institute of Standards and Technology). Such cryptographic standards have significant practical impact because they facilitate the widespread use of sound techniques, and promote interoperability between different implementations.

The remainder of this paper is organized as follows. §2 summarizes the desirable security and performance attributes of a key agreement protocol. In §3, we review the basic ephemeral (short-term) and static (long-term) Diffie–Hellman key agreement protocols and point out their limitations. §4 presents the KEA, Unified Model, and MQV authenticated key agreement protocols, while in §5 we discuss protocols for authenticated key agreement with key confirmation. The protocols are compared in §6. Recent progress in defining and proving the security of key agreement protocols is reviewed in §7. §8 concludes with some directions for future research.

2 Goals of key agreement

This section discusses in more detail the goals of asymmetric authenticated key establishment protocols. The complexity and variety of these goals explains in part the difficulties involved in designing secure protocols.

The fundamental goal of any authenticated key establishment protocol is to distribute keying data. Ideally, the established key should have precisely the same

attributes as a key established face-to-face — for example, it should be shared by the (two) specified entities, it should be distributed uniformly at random from the key space, and no unauthorized (and computationally bounded) entity should learn anything about the key. A protocol achieving this idealistic goal could then be used as a drop-in replacement for face-to-face key establishment without the need to review system security in much the same way as pseudorandom bit generators can replace random bit generators.

Unfortunately, such an abstract goal is not easily attained and it is not an easy task to identify and enunciate the precise security requirements of authenticated key establishment. Nonetheless over the years several concrete security and performance attributes have been identified as desirable. These are informally described in the remainder of this section. Recent more formal attempts at capturing concrete security definitions are discussed in §7.

The first step is to identify what types of attacks it is vital for a protocol to withstand. Since protocols are used over open networks like the Internet, a secure protocol should be able to withstand both *passive* attacks (where an adversary attempts to prevent a protocol from achieving its goals by merely observing honest entities carrying out the protocol) and *active* attacks (where an adversary additionally subverts the communications by injecting, deleting, altering or replaying messages).

The second step is to identify what concrete security goals it is vital for a protocol to provide. The fundamental security goals described below are considered to be vital in any application. The other security and performance attributes are important in some environments, but less important in others.

Fundamental security goals. Let A and B be two honest entities, i.e., legitimate entities who execute the steps of a protocol correctly.

1. *implicit key authentication.* A key agreement protocol is said to provide implicit key authentication (of B to A) if entity A is assured that no other entity aside from a specifically identified second entity B can possibly learn the value of a particular secret key. Note that the property of implicit key authentication does not necessarily mean that A is assured of B actually possessing the key.
2. *explicit key authentication.* A key agreement protocol is said to provide *explicit key confirmation* (of B to A) if entity A is assured that the second entity B *has actually* computed the agreed key. The protocol provides *implicit key confirmation* if A is assured that B *can* compute the agreed key. While explicit key confirmation appears to provide stronger assurances to A than implicit key confirmation (in particular, the former implies the latter), it appears that, for all practical purposes, the assurances are in fact the same. That is, the assurance that A requires in practice is merely that B can compute the key rather than that B has actually computed the key. Indeed in practice, even if a protocol does provide explicit key confirmation, it cannot guarantee to A that B will not lose the key between key establishment and key use. Thus it would indeed seem that implicit key confirmation and

explicit key confirmation are in practice very similar, and the remainder of this paper will not distinguish between the two.
Key confirmation by itself is not a useful service — it is only desirable when accompanied with implicit key authentication. A key agreement protocol is said to provide explicit key authentication (of B to A) if both implicit key authentication and key confirmation (of B to A) are provided.

A key agreement protocol which provides implicit key authentication to both participating entities is called an *authenticated key agreement (AK)* protocol, while one providing explicit key authentication to both participating entities is called an *authenticated key agreement with key confirmation (AKC)* protocol.

Key agreement protocols in which the services of implicit key authentication or explicit key authentication are provided to only one (*unilateral*) rather than both (*mutual*) participating entities are also useful in practice, for example in encryption applications where only authentication of the intended recipient is required. Such unilateral key agreement protocols (e.g., ElGamal key agreement [33, Protocol 12.52]) are not considered in this paper.

Other desirable security attributes. A number of other desirable security attributes have also been identified. Typically the importance of supplying these attributes will depend on the application. In the following, A and B are two honest entities.

1. *known-key security.* Each run of a key agreement protocol between A and B should produce a unique secret key; such keys are called *session* keys. Session keys are desirable in order to limit the amount of data available for cryptanalytic attack (e.g., ciphertext generated using a fixed session key in an encryption application), and to limit exposure in the event of (session) key compromise. A protocol should still achieve its goal in the face of an adversary who has learned some other session keys.
2. *forward secrecy.* If long-term private keys of one or more entities are compromised, the secrecy of previous session keys established by honest entities is not affected. A distinction is sometimes made between the scenario in which a single entity's private key entity is compromised (*half* forward secrecy) and the scenario in which the private keys of both participating entities are compromised (*full* forward secrecy).
3. *key-compromise impersonation.* Suppose A's long-term private key is disclosed. Clearly an adversary that knows this value can now impersonate A, since it is precisely this value that identifies A. However, it may be desirable in some circumstances that this loss does not enable the adversary to impersonate other entities to A.
4. *unknown key-share.* Entity B cannot be coerced into sharing a key with entity A without B's knowledge, i.e., when B believes the key is shared with some entity $C \neq A$, and A (correctly) believes the key is shared with B.
A hypothetical scenario where an unknown key-share attack can have damaging consequences is the following; this scenario was first described by Diffie, van Oorschot and Wiener [21]. Suppose that B is a bank branch and A is an

account holder. Certificates are issued by the bank headquarters and within each certificate is the account information of the holder. Suppose that the protocol for electronic deposit of funds is to exchange a key with a bank branch via a mutually authenticated key agreement. Once B has authenticated the transmitting entity, encrypted funds are deposited to the account number in the certificate. Suppose that no further authentication is done in the encrypted deposit message (which might be the case to save bandwidth). If the attack mentioned above is successfully launched then the deposit will be made to C's account instead of A's account.

Desirable performance attributes. These include:

1. Minimal number of *passes* (the number of messages exchanged).
2. Low *communication overhead* (total number of bits transmitted).
3. Low *computation overhead* (total number of arithmetical operations required).
4. Possibility of *precomputation* (to minimize on-line computational overhead).

Other desirable attributes. These include:

1. *Anonymity* of the entities participating in a run of the protocol.
2. *Role symmetry* (the messages transmitted have the same structure).
3. *Non-interactiveness* (the messages transmitted between the two entities are independent of each other).
4. Non-reliance on encryption in order to meet export restrictions.
5. Non-reliance on hash functions since these are notoriously hard to design.
6. Non-reliance on timestamping since it is difficult to implement securely in practice.

3 Diffie–Hellman key agreement

This section describes the basis of Diffie–Hellman based key agreement protocols and motivates the modern protocols we describe in §4 and §5 by illustrating some of the deficiencies of early protocols.

The mathematical tool commonly used for devising key agreement protocols is the *Diffie–Hellman problem*: given a cyclic group G of prime order n, a generator g of G, and elements g^x, $g^y \in G$ (where $x, y \in_R [1, n-1]$), find g^{xy}. (We use $x \in_R S$ to denote that x is chosen uniformly at random from the set S.) This problem is closely related to the widely-studied *discrete logarithm problem* (given G, n, g, and g^x where $x \in_R [0, n-1]$, find x), and there is strong evidence that the two problems are computationally equivalent (e.g., see [16] and [32]).

For concreteness, this paper deals with the case where G is a prime order subgroup of $\mathbb{Z}_p^*$, the multiplicative group of the integers modulo a prime p. However, the discussion applies equally well to any group of prime order in which the discrete logarithm problem is computationally intractable, for example prime order subgroups of the group of points on an elliptic curve over a finite field. The following notation is used throughout the paper.

A, B	Honest entities.
p	1024-bit prime.
q	160-bit prime divisor of $p-1$.
g	An element of order q in $\mathbb{Z}_p^*$.
a, b	Static private keys of A and B; $a, b \in_R [1, q-1]$.
Y_A, Y_B	Static public keys of A and B; $Y_A = g^a \bmod p$, $Y_B = g^b \bmod p$.
x, y	Ephemeral private keys of A and B; $x, y \in_R [1, q-1]$.
R_A, R_B	Ephemeral public keys of A and B; $R_A = g^x \bmod p$, $R_B = g^y \bmod p$.
H	A cryptographic hash function (e.g., SHA-1 [35]).
MAC	A message authentication code algorithm (e.g., [4,6,7]).

The operator mod p will henceforth be omitted.

The *domain* parameters (p, q, g) are common to all entities. For the remainder of this paper, we will assume that static public keys are exchanged via certificates. Cert_A denotes A's public-key certificate, containing a string of information that uniquely identifies A (such as A's name and address), her static public key Y_A, and a certifying authority CA's signature over this information. Other information may be included in the data portion of the certificate, including the domain parameters if these are not known from context. Any other entity B can use his authentic copy of the CA's public key to verify A's certificate, thereby obtaining an authentic copy of A's static public key.

We assume that the CA has verified that A possess the private key a corresponding to her static public key Y_A. This is done in order to prevent potential unknown key-share attacks whereby an adversary E registers A's public key Y_A as its own and subsequently deceives B into believing that A's messages originated from E (see [15] for more details). Checking knowledge of private keys is in general a sensible precaution and is often vital for theoretical analysis. We also assume that the CA has verified the validity of A's static public key Y_A, i.e., the CA has verified that $1 < Y_A < p$ and that $(Y_A)^q \equiv 1 \pmod p$; this process is called *public key validation* [26]. Rationale for performing public key validation is provided in §4.1.

The first asymmetric key agreement protocol was proposed by Diffie and Hellman in their seminal 1976 paper [20]. We present two versions of the basic protocol, one where the entities exchange *ephemeral* (short-term) public keys, and the other where the entities exchange *static* (long-term) public keys.

Protocol 1 (Ephemeral Diffie–Hellman)

1. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ to B.
2. B selects $y \in_R [1, q-1]$ and sends $R_B = g^y$ to A.
3. A computes $K = (R_B)^x = g^{xy}$.
4. B computes $K = (R_A)^y = g^{xy}$.

While the ephemeral Diffie–Hellman protocol provides implicit key authentication in the presence of passive adversaries, it does not on its own provide any useful services in the presence of active adversaries since neither entity is provided with any assurances regarding the identity of the entity it is communicating with. (See also Table 1 in §6.) This drawback can be overcome by using public keys that have been certified by a trusted CA.

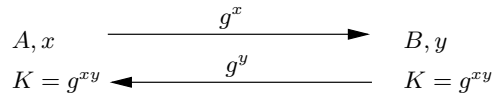

Fig. 1. Protocol 1 (Ephemeral Diffie–Hellman).

Protocol 2 (Static Diffie–Hellman)

1. A sends Cert_A to B.
2. B sends Cert_B to A.
3. A computes $K = (Y_B)^a = g^{ab}$.
4. B computes $K = (Y_A)^b = g^{ab}$.

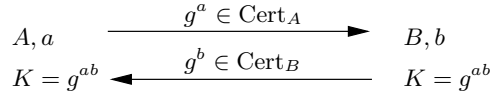

Fig. 2. Protocol 2 (Static Diffie–Hellman).

Since each entity is assured that it possesses an authentic copy of the other entity's public key, the static Diffie–Hellman protocol offers implicit key authentication. A major drawback, however, is that A and B compute the same shared secret $K = g^{ab}$ for each run of the protocol.

The drawbacks of the ephemeral and static Diffie–Hellman protocols can be alleviated by using both static and ephemeral keying material in the formation of shared secrets. An example of an early protocol designed in this manner is the MTI/C0 protocol [31].

Protocol 3 (MTI/C0)

1. A selects $x \in_R [1, q-1]$ and sends $T_A = (Y_B)^x$ to B.
2. B selects $y \in_R [1, q-1]$ and sends $T_B = (Y_A)^y$ to A.
3. A computes $K = (T_B)^{a^{-1}x} = g^{xy}$.
4. B computes $K = (T_A)^{b^{-1}y} = g^{xy}$.

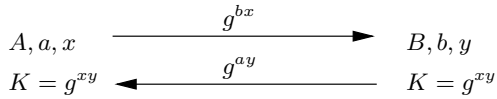

Fig. 3. Protocol 3 (MTI/C0).

This protocol appears secure at first glance. Unfortunately, it turns out that this attempt to combine static and ephemeral Diffie–Hellman protocols has introduced some subtle problems. As an example, consider the following instance of the *small subgroup* attack [29] on the MTI/C0 protocol. An adversary E replaces T_A and T_B with the identity element 1. Both A and B now form $K = 1$, which is also known to E. This attack demonstrates that the MTI/C0 protocol (as described above) does not offer implicit key authentication.

The 3 protocols described in this section demonstrate some of the subtleties involved in designing secure authenticated key agreement protocols. Other kinds of attacks that have been identified besides small subgroup attacks include:

1. *intruder-in-the-middle attack* [37]. In this classic attack on ephemeral Diffie–Hellman, the adversary replaces A's and B's ephemeral keys g^x and g^y with keys $g^{\overline{x}}$ and $g^{\overline{y}}$ of its choice. E can then compute the session keys formed by A and B ($g^{x\overline{y}}$ and $g^{\overline{x}y}$, respectively), and use these to translate messages exchanged between A and B that are encrypted under the session keys.
2. *reflection attack* [34]. A's challenges are replayed back to A as messages purportedly from B.
3. *interleaving attack* [12,21]. The adversary reuses messages transmitted during a run of the protocol in other runs of the protocol.

Such attacks are typically very subtle and require little computational overhead. They highlight the necessity of some kind of formal analysis to avoid the use of flawed protocols.

4 AK protocols

This section discusses some AK protocols currently proposed in standards. We present the two-pass KEA, Unified Model, and MQV protocols, and their one-pass variants.

Before we present the AK protocols it is worth reminding the reader that, as discussed in §2, it is highly desirable for key establishment protocols to provide explicit key authentication. Thus, when AK protocols are used in practice, key confirmation should usually be added to the protocols. Nonetheless it is worth presenting the raw AK protocols since key confirmation can be achieved in a variety of ways and it is sometimes desirable to separate key confirmation from implicit key authentication and move the burden of key confirmation from the key establishment mechanism to the application. For example, if the key is to be subsequently used to achieve confidentiality, then encryption with the key can begin on some (carefully chosen) known data. Other systems may provide key confirmation during a 'real-time' telephone conversation. We present a generic method for securely incorporating key confirmation into AK protocols in §5.

4.1 KEA

The Key Exchange Algorithm (KEA) was designed by the National Security Agency (NSA) and declassified in May 1998 [36]. It is the key agreement protocol

in the FORTEZZA suite of cryptographic algorithms designed by NSA in 1994. KEA is very similar to the Goss [23] and MTI/A0 [31] protocols.

Protocol 4 (KEA)

1. A and B obtain authentic copies of each other's public keys Y_A and Y_B.
2. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ to B.
3. B selects $y \in_R [1, q-1]$ and sends $R_B = g^y$ to A.
4. A verifies that $1 < R_B < p$ and $(R_B)^q \equiv 1 \pmod{p}$. If any check fails, then A terminates the protocol run with failure. Otherwise, A computes the shared secret $K = (Y_B)^x + (R_B)^a \bmod p$. If $K = 0$, then A terminates the protocol run with failure.
5. B verifies that $1 < R_A < p$ and $(R_A)^q \equiv 1 \pmod{p}$. If any check fails, then B terminates the protocol run with failure. Otherwise, B computes the shared secret $K = (Y_A)^y + (R_A)^b \bmod p$. If $K = 0$, then B terminates the protocol run with failure.
6. Both A and B compute the 80-bit session key $k = \mathrm{kdf}(K)$, where kdf is a key derivation function derived from the symmetric-key encryption scheme SKIPJACK (see [36] for further details).

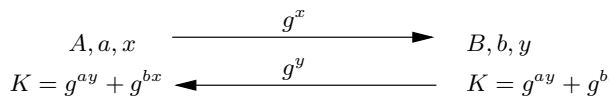

Fig. 4. Protocol 4 (KEA).

To illustrate the need for the features of KEA, we demonstrate how the protocol is weakened when certain modifications are made. This serves to further illustrate that designing secure key agreement protocols is a delicate and difficult task, and that subtle changes to a protocol can render it insecure.

Validation of public keys – verifying that they lie in the subgroup of order q. Suppose that A does not verify that $(R_B)^q \equiv 1 \pmod{p}$. Then, as observed by Lim and Lee [30], it may be possible for a malicious B to learn information about A's static private key a as follows using a variant of the small subgroup attack. Suppose that $p-1$ has a prime factor l of small bitlength (e.g., 40 bits). Let $\beta \in \mathbb{Z}_p^*$ be of order l. If B sends $R_B = \beta$ to A, then A computes $K = g^{bx} + \beta^a \bmod p$ and $k = \mathrm{kdf}(K)$. Suppose now that A sends B an encrypted message $c = E_k(m)$, where E is a symmetric-key encryption scheme and the plaintext m has some recognizable structure. For each d, $0 \leq d \leq l-1$, B computes $K' = g^{bx} + \beta^d \bmod p$, $k' = \mathrm{kdf}(K')$, and $m' = E_k^{-1}(c)$. If m' possesses the requisite structure, then B concludes that $d = a \bmod l$, thus learning some partial information about a. This can be repeated for different small prime factors l of $p-1$.

Validation of public keys – verifying that they lie in the interval $[2, p-1]$. Suppose that A does not verify that $1 < Y_B < p$ and $1 < R_B < p$. Then an adversary E can launch the following unknown key-share attack. E gets $Y_E = 1$ certified as its static public key. E then forwards A's ephemeral public key R_A to B alleging it came from E. After B replies to E with R_B, E sends $R'_B = 1$ to A alleging it came from B. A computes $K_{AB} = g^{bx} + 1$ and B computes $K_{BE} = g^{bx} + 1$. Thus B is coerced into sharing a key with A without B's knowledge.

Use of a key derivation function. The key derivation function kdf is used to derive a session key from the shared secret key K. One reason for doing this is to mix together *strong* bits and potential *weak* bits of K — weak bits are certain bits of information about K that can be correctly predicted with non-negligible advantage.

Another reason is to destroy the algebraic relationships between the shared secret K and the static and ephemeral public keys. This can help prevent against some kinds of known-key attacks, such as Burmester's triangle attack [17] which we describe next. An adversary E, whose static key pair is (c, g^c), observes a run of protocol between A and B in which ephemeral public keys g^x and g^y are exchanged; the resulting shared secret is $K_{AB} = g^{ay} + g^{bx}$. E then initiates a run of the protocol with A, replaying g^y as its ephemeral public key; the resulting secret which only A can compute is $K_{AE} = g^{ay} + g^{c\overline{x}}$, where $g^{\overline{x}}$ is A's ephemeral public key. Similarly, E initiates a run of the protocol with B, replaying g^x as its ephemeral public key; the resulting secret which only B can compute is $K_{BE} = g^{bx} + g^{c\overline{y}}$, where $g^{\overline{y}}$ is B's ephemeral public key. If E can somehow learn K_{AE} and K_{BE} (this is the known-key portion of the attack), then E can compute $K_{AB} = K_{AE} + K_{BE} - g^{c\overline{x}} - g^{c\overline{y}}$.

The check that $K \neq 0$. This check is actually unnecessary as the following argument shows. Since $(g^b)^q \equiv (g^y)^q \equiv 1 \pmod{p}$, we have that $(g^{bx})^q \equiv (g^{ay})^q \equiv 1 \pmod{p}$. Now, $K = 0$ if and only if $g^{bx} \equiv -g^{ay} \pmod{p}$. But this is impossible since otherwise $(g^{bx})^q \equiv (-g^{ay})^q \equiv (-1)^q \equiv -1 \pmod{p}$.

Security notes. KEA does not provide (full) forward secrecy since an adversary who learns a and b can compute all session keys established by A and B. See also Table 1 in §6.

4.2 The Unified Model

The Unified Model, proposed by Ankney, Johnson and Matyas [1], is an AK protocol that is in the draft standards ANSI X9.42 [2], ANSI X9.63 [3], and IEEE P1363 [24]. One of its advantages is that it is conceptually simple and consequently easier to analyze (see §7.1).

Protocol 5 (Unified Model)

1. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ and Cert_A to B.
2. B selects $y \in_R [1, q-1]$ and sends $R_B = g^y$ and Cert_B to A.

3. A verifies that $1 < R_B < p$ and $(R_B)^q \equiv 1 \pmod{p}$. If any check fails, then A terminates the protocol run with failure. Otherwise, A computes the session key $k = H((Y_B)^a \| (R_B)^x)$.
4. B verifies that $1 < R_A < p$ and $(R_A)^q \equiv 1 \pmod{p}$. If any check fails, then B terminates the protocol run with failure. Otherwise, B computes the session key $k = H((Y_A)^b \| (R_A)^y)$.

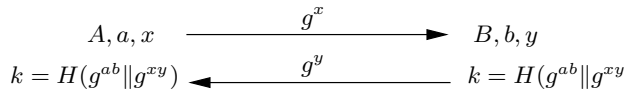

Fig. 5. Protocol 5 (Unified model).

SECURITY NOTES. The Unified Model does not provide the service of key compromise impersonation, since an adversary who learns a can impersonate any other entity B to A. See also Table 1 in §6.

4.3 MQV

The so-called MQV protocol [29] is an AK protocol that is in the draft standards ANSI X9.42 [2], ANSI X9.63 [3], and IEEE P1363 [24]. The following notation is used. If $X \in [1, p-1]$, then $\overline{X} = (X \bmod 2^{80}) + 2^{80}$; more generally, $\overline{X} = (X \bmod 2^{\lceil f/2 \rceil}) + 2^{\lceil f/2 \rceil}$, where f is the bitlength of q. Note that $(\overline{X} \bmod q) \neq 0$.

Protocol 6 (MQV)

1. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ and Cert_A to B.
2. B selects $y \in_R [1, q-1]$ and sends $R_B = g^y$ and Cert_B to A.
3. A verifies that $1 < R_B < p$ and $(R_B)^q \equiv 1 \pmod{p}$. If any check fails, then A terminates the protocol run with failure. Otherwise, A computes $s_A = (x + a\overline{R}_A) \bmod q$ and the shared secret $K = (R_B(Y_B)^{\overline{R}_B})^{s_A}$. If $K = 1$, then A terminates the protocol run with failure.
4. B verifies that $1 < R_A < p$ and $(R_A)^q \equiv 1 \pmod{p}$. If any check fails, then B terminates the protocol run with failure. Otherwise, B computes $s_B = (y + b\overline{R}_B) \bmod q$ and the shared secret $K = (R_A(Y_A)^{\overline{R}_A})^{s_B}$. If $K = 1$, then B terminates the protocol run with failure.
5. The session key is $k = H(K)$.

SECURITY NOTES. The expression for $\overline{R}_A$ uses only half the bits of R_A. This was done in order to increase the efficiency of computing K because the modular exponentiation $(Y_A)^{\overline{R}_A}$ can be done in half the time of a full exponentiation. The modification does not appear to affect the security of the protocol. The definition of $\overline{R}_A$ implies that $\overline{R}_A \neq 0$; this ensures that the contribution of the static private key a is not being cancelled in the formation of s_A.

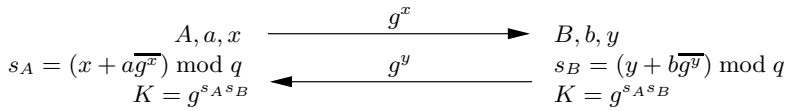

Fig. 6. Protocol 6 (MQV).

The check $K = 1$ ensures that K has order q.

Kaliski [27] has recently observed that Protocol 6 does not possess the unknown key-share attribute. This is demonstrated by the following on-line attack. An adversary E intercepts A's ephemeral public key R_A intended for B, and computes $R_E = R_A(Y_A)^{\overline{R}_A} g^{-1}$, $e = (\overline{R}_E)^{-1} \bmod q$, and $Y_E = g^e$. E then gets Y_E certified as her static public key (note that E knows the corresponding private key e), and transmits R_E to B. B responds by sending R_B to E, which E forwards to A. Both A and B compute the same session key k, however B mistakenly believes that he shares k with E. We emphasize that lack of the unknown key-share attribute does not contradict the fundamental goal of mutual implicit key authentication — by definition the provision of implicit key authentication is only considered in the case where B engages in the protocol with an honest entity (which E isn't). If an application using Protocol 6 is concerned with the lack of the unknown key-share attribute under such on-line attacks, then appropriate key confirmation should be added, for example as specified in Protocol 8 in §5.

4.4 One-pass variants

The purpose of a one-pass AK protocol is for entities A and B to agree upon a session key by only having to transmit one message from A to B — this assumes that A a priori has an authentic copy of B's static public key. One-pass protocols can be useful in applications where only one entity is on-line, such as secure email. Their main security drawbacks are that they do not offer known-key security (since an adversary can replay A's ephemeral public key to B) and forward secrecy (since entity B does not contribute a random per-message component).

The 3 two-pass AK protocols (KEA, Unified Model, MQV) presented in this section can be converted to one-pass AK protocols by simply setting B's ephemeral public key equal to his static public key. We illustrate this next for the one-pass variant of the MQV protocol. A summary of the security services of the 3 one-pass variants is provided in Table 1 in §6.

Protocol 7 (One-pass MQV)

1. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ and Cert_A to B.
2. A computes $s_A = (x + a\overline{R}_A) \bmod q$ and the shared secret $K = (Y_B(Y_B)^{\overline{Y}_B})^{s_A}$. If $K = 1$, then A terminates the protocol run with failure.

3. B verifies that $1 < R_A < p$ and $(R_A)^q \equiv 1 \pmod{p}$. If any check fails, then B terminates the protocol run with failure. Otherwise, B computes $s_B = (b + b\overline{Y}_B) \bmod q$ and the shared secret $K = (R_A(Y_A)^{\overline{R}_A})^{s_B}$. If $K = 1$, then B terminates the protocol run with failure.
4. The session key is $k = H(K)$.

$$\begin{array}{rcl} A, a, x & \xrightarrow{\quad g^x \quad} & B, b \\ s_A = (x + a\overline{g^x}) \bmod q & & s_B = (b + b\overline{g^b}) \bmod q \\ K = g^{s_A s_B} & & K = g^{s_A s_B} \end{array}$$

Fig. 7. Protocol 7 (One-pass MQV).

5 AKC protocols

This section discusses AKC protocols and describes a method to derive AKC protocols from AK protocols.

The following three-pass AKC protocol [13] is derived from the Unified Model AK protocol (Protocol 5) by adding the MACs of the flow number, identities, and the ephemeral public keys. Here, H_1 and H_2 are 'independent' hash functions. In practice, one may choose $H_1(m) = H(10, m)$ and $H_2(m) = H(01, m)$, where H is a cryptographic hash function.

The MACs are computed under the shared key k', which is different from the session key k; Protocol 8 thus offers *implicit* key confirmation. If *explicit* key confirmation were to be provided by using the session key k as the MAC key, then a passive adversary would learn some information about k — the MAC of a known message under k. The adversary can use this to distinguish k from a key selected uniformly at random from the key space. This variant therefore sacrifices the desired goal that a protocol establish a computationally indistinguishable key. The maxim that a key establishment protocol can be used as a drop-in replacement for face-to-face key establishment therefore no longer applies and in theory security must be analyzed on a case-by-case basis. We therefore prefer Protocol 8.

Protocol 8 (Unified Model with key confirmation)

1. A selects $x \in_R [1, q-1]$ and sends $R_A = g^x$ and Cert_A to B.
2. (a) B verifies that $1 < R_A < p$ and $(R_A)^q \equiv 1 \pmod{p}$. If any check fails, then B terminates the protocol run with failure.
   (b) B selects $y \in_R [1, q-1]$, and computes $R_B = g^y$, $k' = H_1((Y_A)^b \| (R_A)^y)$, $k = H_2((Y_A)^b \| (R_A)^y)$, and $m_B = \mathrm{MAC}_{k'}(2, B, A, R_B, R_A)$.
   (c) B sends R_B, Cert_B, and m_B to A.
3. (a) A verifies that $1 < R_B < p$ and $(R_B)^q \equiv 1 \pmod{p}$. If any check fails, then A terminates the protocol run with failure.

(b) A computes $k' = H_1((Y_B)^a \| (R_B)^x)$ and $m'_B = \mathrm{MAC}_{k'}(2, B, A, R_B, R_A)$, and verifies $m'_B = m_B$.
(c) A computes $m_A = \mathrm{MAC}_{k'}(3, A, B, R_A, R_B)$ and $k = H_2((Y_B)^a \| (R_B)^x)$, and sends R_A and m_A to B.

4. B computes $m'_A = \mathrm{MAC}_{k'}(3, A, B, R_A, R_B)$ and verifies that $m'_A = m_A$.
5. The session key is k.

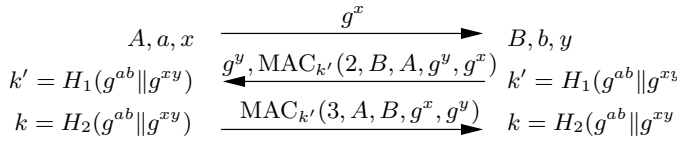

Fig. 8. Protocol 8 (Unified model with key confirmation).

In a similar manner, one can derive three-pass AKC protocols from the KEA (*KEA with key confirmation*) and MQV (*MQV with key confirmation*) AK protocols. The AKC variants of the Unified Model and MQV protocols are being considered for inclusion in ANSI X9.63 [3].

A summary of the security services provided by the 3 AKC variants is given in Table 1 in §6. This table illustrates why AKC protocols may be preferred over AK protocols in practice. First, the incorporation of key confirmation may provide additional security attributes which are not present in the AK protocol. For example, addition of key confirmation in the manner described above makes the MQV protocol resistant to unknown key-share attacks. Second, the security properties of AKC protocols appear to be better understood; see also the discussion in §7.1. Note that since the MACs can be computed efficiently, this method of adding key confirmation to an AK protocol does not place a significant computational burden on the key establishment mechanism.

6 Comparison

This section compares the security and efficiency of the protocols presented in §4 and §5.

Security services. Table 1 contains a summary of the services that are believed to be provided by the AK and AKC protocols discussed in §4 and §5. Although only implicit and explicit key authentication are considered vital properties of key establishment, any new results related to other information in this table would be interesting.

The services are discussed in the context of an entity A who has successfully executed the key agreement protocol over an open network wishing to establish keying data with entity B. In the table:

Scheme	IKA	EKA	K-KS	FS	K-CI	UK-S
Ephemeral Diffie–Hellman	×	×	$\surd?^a$	n/a	n/a	×
Ephemeral Diffie–Hellman (against passive attack)	$\surd\surd$	×	$\surd\surd$	n/a	n/a	$\surd\surd$
Static Diffie–Hellman	$\surd\surd$	×	×	×	×	$\surd\surd$
One-pass KEA	$\surd\surd$	×	×	×	$\surd I$	$\surd\surd$
KEA	$\surd\surd$	×	$\surd\surd$	×	$\surd\surd$	$\surd\surd$
KEA with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	×	$\surd\surd$	$\surd\surd$
One-pass Unified Model	$\surd\surd$	×	×	×	$\surd I$	$\surd\surd$
Unified Model	$\surd\surd$	×	$\surd?^a$	$\surd?^b$	×	$\surd\surd$
Unified Model with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	×	$\surd\surd$
One-pass MQV	$\surd\surd$	×	×	×	$\surd I$	×
MQV	$\surd\surd$	×	$\surd\surd$	$\surd?^b$	$\surd\surd$	×
MQV with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$

[a] Here the technicality hinges on the definition of what contributes 'another session key'. The service of known-key security is certainly provided if the protocol is extended so that explicit authentication of all session keys is supplied.

[b] Again the technicality concerns key confirmation. Both protocols provide forward secrecy if explicit authentication is supplied for all session keys. If not supplied, then the service of forward secrecy cannot be guaranteed.

Table 1. Security services offered by authenticated key agreement protocols.

- $\surd\surd$ indicates that the assurance is provided to A no matter whether A initiated the protocol or not.
- $\surd?$ indicates that the assurance is provided modulo a theoretical technicality.
- $\surd$I indicates that the assurance is provided to A only if A is the protocol's initiator.
- × indicates that the assurance is not provided to A by the protocol.

The names of the services have been abbreviated to save space: IKA denotes implicit key authentication, EKA explicit key authentication, K-KS known-key security, FS forward secrecy, K-CI key-compromise impersonation, and UK-S unknown key-share.

The provision of these assurances is considered in the case that both A and B are honest and have always executed the protocol correctly. The requirement that A and B are honest is certainly necessary for the provision of any service by a key establishment protocol: no key establishment protocol can protect against a dishonest entity who chooses to reveal the session key... just as no encryption scheme can guard against an entity who chooses to reveal confidential data.

Efficiency. The work done by each entity is dominated by the time to perform the modular exponentiations. The total number of modular exponentiations per entity for the KEA, Unified Model, and MQV AK protocols is 4, 4, and 3.5,

respectively. If precomputations (of quantities involving the entity's static and ephemeral keys and the other entity's static keys) are discounted, then the total number of on-line modular exponentiations per entity reduces to 2, 2, and 2.5, respectively.

As noted in §5, MACs can be computed efficiently and hence the AKC variants have essentially the same computational overhead as their AK counterparts. They do, however, require an extra flow.

7 Provable security

This section discusses methods that have been used to formally analyze key agreement protocols. The goal of these methods is to facilitate the design of secure protocols that avoid subtle flaws like those described in §3. We examine two approaches, provable security and formal methods, focusing on the former.

Provable security was invented in the 1980's and applied to encryption schemes and signature schemes. The process of proving security of a protocol comes in five stages:

1. Specification of model.
2. Definition of goals within this model.
3. Statement of assumptions.
4. Description of protocols.
5. Proof that the protocol meets its goals within the model.

As discussed in §1, the emphasis of work in provable security of a protocol should be how appropriate the model, definitions, and underlying assumptions are, rather than the mere statement that a protocol attains provable security — after all, all protocols are provably secure in some model, under some definitions, or under some assumptions.

History of provable security. Building on earlier informal work of Bird et al. [12] for the symmetric setting and Diffie, van Oorschot and Wiener [21] for the asymmetric setting, Bellare and Rogaway [9] provided a model of distributed computing and rigorous security definitions, proposed concrete two-party authenticated key transport protocols in the symmetric setting, and proved them secure under the assumption that a pseudorandom function family exists. They then extended the model to handle the three-party (*Kerberos*) case [10]; see also Shoup and Rubin [39] for an extension of this work to the smart card world. Blake-Wilson and Menezes [14] and Blake-Wilson, Johnson and Menezes [13] extended the Bellare-Rogaway model to the asymmetric setting, and proposed and proved the security of some authenticated key transport, AK, and AKC protocols (see §7.1). More recently, Bellare, Canetti and Krawczyk [5] provided a systematic method for transforming authentication protocols that are secure in a model of idealized authenticated communications into protocols that are secure against active attacks; their work is discussed further in §7.2.

Formal methods. These are methods for analyzing cryptographic protocols in which the communications system is described using a formal specification language which has some mathematical basis, from which security properties of the protocol can be inferred. (See [38] and [28] for surveys on formal methods.) The most widely used of these methods are those related to the *BAN logic* of Burrows, Abadi and Needham [18], which was extended by van Oorschot [40] to enable the formal analysis of authenticated key agreement protocols in the asymmetric setting. Such methods begin with a set of beliefs for the participants and use logical inference rules to derive a belief that the protocol goals have been obtained.

Such formal methods have been useful in uncovering flaws and redundancies in protocols. However, they suffer from a number of shortcomings when considered as tools for designing high-assurance protocols. First, a proof that a protocol is logically correct does not imply that it is secure. This is especially the case because the process of converting a protocol into a formal specification may itself be subject to subtle flaws. Second, there is no clear security model associated with the formal systems used and thus it is hard to assess whether the implied threat model corresponds with the requirements of an application. Therefore, we believe that provable security techniques offer greater assurance than formal methods and we focus on provable security for the remainder of this section.

7.1 Bellare-Rogaway model of distributed computing

Work on the design of provably secure authenticated key agreement has largely focused on the Bellare-Rogaway model of distributed computing [9,10].

The Bellare-Rogaway model, depicted in Figure 9, is a formal model of communication over an open network in which the adversary E is afforded enormous power. She controls all communication between entities, and can at any time ask an entity to reveal its static private key. Furthermore, she may at any time initiate sessions between any two entities, engage in multiple sessions with the same entity at the same time, and ask an entity to enter a session with itself. We provide an informal description of the Bellare-Rogaway model, and informal definitions of the goals of secure AK and AKC protocols. For complete descriptions, see [9,10,13].

In the model, E is equipped with a collection of $\Pi^s_{A,B}$ oracles. $\Pi^s_{A,B}$ models entity A who believe she is communicating with entity B for the s^{th} time. E is allowed to make three types of queries of its oracles:

`Send`$(\Pi_{A,B}, x)$: E gives a particular oracle x as input and learns the oracle's response.
`Reveal`$(\Pi_{A,B})$: E learns the session key (if any) the oracle currently holds.
`Corrupt`(A): E learns A's static private key.

When E asks an oracle a query, the oracle computes its response using the description of the protocol. Security goals are defined in the context of running E in the presence of these oracles.

Secure key agreement is now captured by a test involving an additional `Test` query. At the end of its experiment, E selects a *fresh* oracle $\Pi^s_{A,B}$ — this is an oracle which has accepted a session key k, and where the adversary has not learned k by trivial means (either by corrupting A or B, or by issuing a `Reveal` query to $\Pi^s_{A,B}$ or to any $\Pi^t_{B,A}$ oracle which has had a matching conversation with $\Pi^s_{A,B}$) — and asks it `Test` query. The oracle replies with either its session key k or a random key, and the adversary's job is to decide which key it has been given.

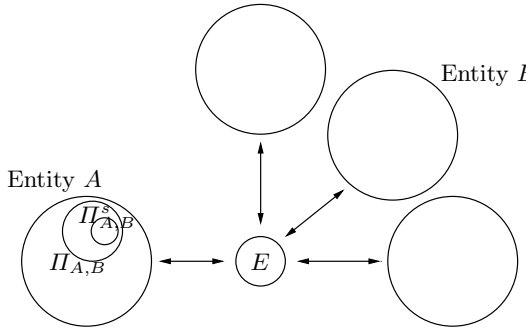

Fig. 9. The Bellare-Rogaway model of distributed computing.

Definition 1 ([13]). *(Informal) An AK protocol is* secure *if:*

(i) The protocol successfully distributes keys in the absence of an adversary.
(ii) No adversary E can distinguish a session key held by a fresh $\Pi^s_{A,B}$ oracle from a key selected uniformly at random.

A secure AKC protocol is defined by amalgamating the notion of entity authentication with the notion of a secure AK protocol.

Definition 2 ([13]). *(Informal) An AKC protocol is* secure *if in addition to conditions (i) and (ii) of Definition 1:*

(iii) The only way an adversary E can induce a $\Pi^s_{A,B}$ oracle to accept a session key is by honestly transmitting messages between $\Pi^s_{A,B}$ and some $\Pi^t_{B,A}$.

The security of the Unified Model (Protocol 5) and the Unified Model with key confirmation (Protocol 8) in the Bellare-Rogaway model was proven under certain assumptions in [13].

Theorem 1 ([13]). *Protocol 5 is a secure AK protocol in the Bellare-Rogaway model provided that:*

(i) the adversary makes no `Reveal` queries;

(ii) the Diffie–Hellman problem is hard; and
(iii) H is a random oracle.

Theorem 2 ([13]). *Protocol 8 is a secure AKC protocol in the Bellare-Rogaway model provided that:*

(i) the Diffie–Hellman problem is hard;
(ii) the MAC *is secure; and*
(iii) H_1 and H_2 are independent random oracles.

A random oracle is a 'black-box' random function which is supplied to all entities, including the adversary. The assumption that H, H_1 and H_2 are random oracles is a very powerful one and facilitates security analysis. This so-called *random oracle model* was introduced and popularized by Bellare and Rogaway [8]. In practice, the random oracles can be instantiated with hash functions — therefore the security proofs in the random model are no longer valid in the practical implementation. Nonetheless, and despite recent results demonstrating the limitations of the random oracle model [19], it is a thesis that protocols proven secure in the random oracle provide higher security assurances than protocols deemed secure by ad-hoc means.

To see that Protocol 5 is not a secure AK protocol in the Bellare-Rogaway model if the adversary is allowed to make `Reveal` queries, consider the following interleaving/reflection attack. Suppose that A initiates 2 runs of the protocol; let A's ephemeral public keys be g^x and $g^{\overline{x}}$ in the first and second runs, respectively. The adversary E then replays $g^{\overline{x}}$ and g^x to A in the first and second rounds respectively, purportedly as B's ephemeral public keys. A computes both session keys as $k = H(g^{ab} \| g^{x\overline{x}})$. E can now `Reveal` one session key, and thus also learn the other.

It is conjectured in [13] that the modification of Protocol 5 in which the session key is formed as $k = H(g^{ay} \| g^{bx})$ is a secure AK protocol assuming only that the Diffie–Hellman problem is hard and that H is a random oracle.

7.2 A modular approach

Recently, Bellare, Canetti and Krawczyk [5] have suggested an approach to the design of provably secure key agreement protocols that differs from the Bellare-Rogaway model. Their approach is a modular approach and starts with protocols that are secure in a model of idealized authenticated communication and then systematically transforms them into protocols which are secure in the realistic unauthenticated setting. This approach has the advantage that a new proof of security is not required for each protocol — instead once the approach is justified it can be applied to any protocol that works in the ideal model. On the other hand, it is less clear what practical guarantees are provided so the evaluation of whether the guarantees are appropriate in an application is perhaps less understood. The following is an informal overview of their approach.

AUTHENTICATORS. Authenticators are key to the systematic transformations at the heart of the modular approach. They are compilers that take as input a protocol designed for authenticated networks, and transforms it into an 'equivalent' protocol for unauthenticated networks. The notion of *equivalence* or *emulation* is formalized as follows. A protocol P' designed for unauthenticated networks is said to *emulate* a protocol P designed for authenticated networks, if for each adversary E' of P' there exists an adversary E of P such that for all inputs x, the views $V_{P,E}(x)$ and $V_{P',E'}(x)$ are computationally indistinguishable. (The *view* $V_{P,E}(x)$ of a protocol P which is run on input x in the presence of an adversary E is the random variable describing the cumulative outputs of E and all the legitimate entities.)

MT-AUTHENTICATORS. In [5], authenticators are realized using the simpler idea of an *MT-authenticator* which emulates the most straightforward *message transmission (MT)* protocol in which a single message is passed from A to B as depicted in Figure 10. Figure 11 illustrates the protocol λ_{sig} which is proven in [5] to be an MT-authenticator. In the figure, $\text{sign}_A()$ denotes A's signature using a signature scheme that is secure against chosen message attacks (e.g., [11,22]). Now an MT-authenticator λ can be used to construct a *compiler* C_λ as follows: given a protocol P, $P' = C_\lambda(P)$ is the protocol obtained by applying λ to each message transmitted by P. It is proven in [5] that C_λ is indeed an authenticator.

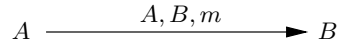

Fig. 10. Message transmission protocol (MT).

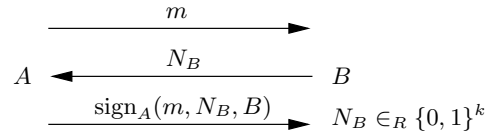

Fig. 11. MT-authenticator λ_{sig}.

KEY ESTABLISHMENT. Finally, this MT-authenticator is used to build a secure authenticated key agreement protocol. It is first shown in [5] that ephemeral Diffie–Hellman EDH (Protocol 1) is a secure key establishment protocol for authenticated networks by showing that it emulates traditional face-to-face key establishment as described in §2. Then, EDH is emulated using $C_{\lambda_{\text{sig}}}$. The result $C_{\lambda_{\text{sig}}}(EDH)$ is a secure six-pass authenticated key agreement protocol. Combining messages from different flows, and replacing the challenges N_A and N_B with

the ephemeral public keys g^x and g^y, respectively, yields the three-pass *BCK protocol*, depicted in Figure 12.

The BCK protocol is similar to Key Agreement Mechanism 7 in ISO/IEC 11770-3 [25]. In the latter, the MACs of the signatures under the shared secret $K = g^{xy}$ are also included in flows 2 and 3, thus providing *explicit* key confirmation, instead of just *implicit* key confirmation as provided by the BCK protocol.

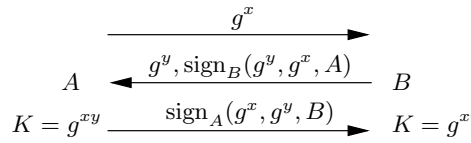

Fig. 12. BCK protocol.

8 Conclusions and future work

This paper surveyed practical and provable security aspects of some authenticated Diffie–Hellman key agreement protocols that are being considered for standardization.

A number of questions can be asked. Can the MQV protocol be proven secure in a reasonable model of computing? Are the definitions of secure AK and AKC protocols in §7.1 the right ones? How do the models and security definitions presented in §7.1 and §7.2 compare? Are the security proofs meaningful in practice? That is, can the reductions used in the proofs be untilized to obtain meaningful measures of exact security [11]? (*Exact security* is a concrete quantification of the security guaranteed by a protocol in terms of the perceived security of the underlying cryptographic primitives, e.g., the Diffie–Hellman problem or a secure MAC algorithm.)

Two important tasks that remain are to devise a provably secure two-pass AK protocol, and to provide formal definitions for secure one-pass key agreement protocols.

References

1. R. Ankney, D. Johnson and M. Matyas, "The Unified Model", contribution to X9F1, October 1995.
2. ANSI X9.42, *Agreement of Symmetric Algorithm Keys Using Diffie–Hellman*, working draft, May 1998.
3. ANSI X9.63, *Elliptic Curve Key Agreement and Key Transport Protocols*, working draft, July 1998.
4. M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication", *Advances in Cryptology – Crypto '96*, LNCS **1109**, 1996, 1-15.

5. M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols", *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`
6. M. Bellare, R. Guerin and P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions", *Advances in Cryptology – Crypto '95*, LNCS **963**, 1995, 15-28.
7. M. Bellare, J. Kilian and P. Rogaway, "The security of cipher block chaining", *Advances in Cryptology – Crypto '94*, LNCS **839**, Springer-Verlag, 1994, 341-358.
8. M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols", *1st ACM Conference on Computer and Communications Security*, 1993, 62-73. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`
9. M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology – Crypto '93*, LNCS **773**, 1994, 232-249. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`
10. M. Bellare and P. Rogaway, "Provably secure session key distribution — the three party case", *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, 1995, 57-66.
11. M. Bellare and P. Rogaway, "The exact security of digital signatures — how to sign with RSA and Rabin", *Advances in Cryptology – Eurocrypt '96*, LNCS **1070**, 1996, 399-416. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`
12. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic design of two-party authentication protocols", *Advances in Cryptology – Crypto '91*, LNCS **576**, 1992, 44-61.
13. S. Blake-Wilson, D. Johnson and A. Menezes, "Key agreement protocols and their security analysis", *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, LNCS **1355**, 1997, 30-45. A full version of this paper is available at `http://www.cacr.math.uwaterloo.ca`
14. S. Blake-Wilson and A. Menezes, "Entity authentication and authenticated key transport protocols employing asymmetric techniques", *Proceedings of the 5th International Workshop on Security Protocols*, LNCS **1361**, 1997, 137-158.
15. S. Blake-Wilson and A. Menezes, "Unknown key-share attacks on the station-to-station (STS) protocol", Technical report CORR 98-42, University of Waterloo, 1998. Also available at `http://www.cacr.math.uwaterloo.ca/`
16. D. Boneh and R. Lipton, "Algorithms for black-box fields and their application to cryptography", *Advances in Cryptology – Crypto '96*, LNCS **1109**, 1996, 283-297.
17. M. Burmester, "On the risk of opening distributed keys", *Advances in Cryptology – Crypto '94*, LNCS **839**, 1994, 308-317.
18. M. Burrows, M. Abadi and R. Needham, "A logic of authentication", *ACM Transactions on Computer Systems*, **8** (1990), 18-36.
19. R. Canetti, O. Goldreich and S. Halevi, "The random oracle methodology, revisited", *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998.
20. W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, **22** (1976), 644-654.
21. W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, **2** (1992), 107-125.
22. C. Dwork and M. Naor, "An efficient existentially unforgeable signature scheme and its applications", *Journal of Cryptology*, **11** (1998), 187-208.

23. K.C. Goss, "Cryptographic method and apparatus for public key exchange with authentication", U.S. patent 4,956,865, September 11 1990.
24. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, working draft, July 1998.
25. ISO/IEC 11770-3, *Information Technology – Security Techniques – Key Management – Part 3: Mechanisms Using Asymmetric Techniques*, draft, (DIS), 1996.
26. D. Johnson, Contribution to ANSI X9F1 working group, 1997.
27. B. Kaliski, Contribution to ANSI X9F1 and IEEE P1363 working groups, June 1998.
28. R. Kemmerer, C. Meadows and J. Millen, "Three systems for cryptographic protocol analysis", *Journal of Cryptology*, **7** (1994), 79-130.
29. L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", Technical report CORR 98-05, University of Waterloo, 1998. Also available at `http://www.cacr.math.uwaterloo.ca/`
30. C. Lim and P. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup", *Advances in Cryptology – Crypto '97*, LNCS **1294**, 1997, 249-263.
31. T. Matsumoto, Y. Takashima and H. Imai, "On seeking smart public-key distribution systems", *The Transactions of the IECE of Japan*, **E69** (1986), 99-106.
32. U. Maurer and S. Wolf, "Diffie–Hellman oracles", *Advances in Cryptology – Crypto '96*, LNCS **1109**, 1996, 283-297.
33. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
34. C. Mitchell, "Limitations of challenge-response entity authentication", *Electronics Letters*, **25** (August 17, 1989), 1195-1196.
35. National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180-1, April 1995.
36. National Security Agency, "SKIPJACK and KEA algorithm specification", Version 2.0, May 29 1998. Also available at `http://csrc.nist.gov/encryption/skipjack-kea.htm`
37. R. Rivest and A. Shamir, "How to expose an eavesdropper", *Communications of the ACM*, **27** (1984), 393-395.
38. A. Rubin and P. Honeyman, "Formal methods for the analysis of authentication protocols", CITI Technical Report 93-7, Information Technology Division, University of Michigan, 1993. Also available at `http://cs.nyu.edu/~rubin/`
39. V. Shoup and A. Rubin, "Session key distribution using smart cards", *Advances in Cryptology – Eurocrypt '96*, LNCS **1070**, 1996, 321-331.
40. P. van Oorschot, "Extending cryptographic logics of belief to key agreement protocols", *1st ACM Conference on Computer and Communications Security*, 1993, 232-243.