Key Agreement in Ad-hoc Networks ⋆

N. Asokan, Philip Ginzboorg

Communication Systems Laboratory, Nokia Research Center,
P.O.Box 407, FIN-00045 NOKIA GROUP, Finland
{n.asokan,philip.ginzboorg}@nokia.com

Abstract

We encounter new types of security problems in ad-hoc networks because such networks have little or no support infrastructure. In this paper we consider one such problem: A group of people in a meeting room do not have access to public key infrastructure or third party key management service, and they do not share any other prior electronic context. How can they set up a secure session among their computers? We examine various alternatives and propose new protocols for password-based multi-party key agreement in this scenario. Our protocols may be applicable in other scenarios, too. We also present a fault-tolerant version of a multi-party Diffie-Hellman key agreement protocol which can be of independent interest.

Keywords: ad-hoc network, key agreement, password authentication.

1 Introduction

1.1 A new key agreement scenario

Consider a small group of people at a conference coming together in a room for an ad-hoc meeting. They would like to set up a wireless network session among their laptop computers for the duration of the meeting. They want to share information securely so that no one outside the room can eavesdrop in on their session. The people physically present in the room know and trust one another personally. However they do not have any *a priori* means of digitally identifying and authenticating one another, such as shared secrets or

⋆ Expanded version of a talk given at the Nordsec '99 workshop in Kista, Sweden, November 1999.

mutually verifiable public key certificate chains or access to trusted key distribution centers. An attacker can monitor and modify all traffic on the wireless communication channel and may also attempt to send messages purporting to come from those who are inside the room. There is no secure communication channel to connect the computers. The problem is: How can the group set up a secure session among their computers under these conditions? Perhaps the first obvious solution that comes to mind is to let the participants pick a new shared secret and use it as an encryption key to secure their communications. As we shall see in Section 1.3, this is unsatisfactory because it involves a trade-off between security and ease of use.

1.2 Ad-hoc networks

The network in the scenario described above is an example of an *ad-hoc network* in which entities construct a communication network with little or no support infrastructure. Three types of infrastructure are relevant to us. The first type is the routing infrastructure in the form of fixed routers and stable links between them. The second type is the server infrastructure of on-line servers which provide various services such as name service, directory services, and trusted third party services. The third type is the organizational and administrative support such as registration of users, issuing of certificates, and cross-certification agreements between different user domains.

Lack of infrastructure introduces problems in routing, name resolution, service discovery, and security that are typically not encountered in networks with full support infrastructure. In the case of security, there is a second source of difficulty. For example, consider the usual form of confidentiality requirement. Participant A requires the following: "only L can read the messages I send", where 'L' is a label in some name space that is meaningful to all participants. A trusts a certification authority CA to correctly certify public encryption keys of entities. Participant A then achieves its confidentiality requirement by encrypting its messages using a public key for L certified by CA. This process started from a certain *prior context*, consisting of the well defined name space, A's confidentiality requirement, and A's trust in CA. In providing security services, one always starts from such a context.[1] But in our scenario, we encounter a new type of prior context. The security requirement is expressed in terms *location* instead of name space: "Only people present in this meeting room can read the messages I send." Thus, the usual confidentiality techniques such as the use of public keys that are certified to belong to a certain name are not relevant.

[1] Security techniques can only help in transforming and transferring the trust assumptions in the prior context. They cannot *create* trust.

A simple way to implement location-based key agreement is to map the location to a set of labels and then use identity-based mechanisms for key agreement. For example, participants could write their IP addresses in a sheet of paper and pass it around. Then, a certificate-based key agreement mechanism like Internet Key Exchange [1] can be used. These public key certificates can allow participants to verify the binding between the IP addresses and keys of other participants. Although this procedure seems like a reasonable solution for our location-based key agreement, even without access to on-line servers, there are two problems with it. First, we cannot know if a certificate presented by a participant has been revoked. Second, participants may be divided between two or more certification hierarchies that do not have cross-certification agreements.

There are other obvious solutions to the problem if additional assumptions are made. For example, we can use a trusted third party capable of determining the locations of players (e.g., the ActiveBadge system [2]). Such support may be available in some ad-hoc networks. But it is too strong an assumption in general. Another solution is to use a physically secure channel limited to those present in the room (for example, infrared channels, or an ad-hoc wired Ethernet) to negotiate the session key before switching to the insecure wireless channel.

In summary, two features of ad-hoc networks prevent the straight forward application of standard techniques to the key agreement problem: lack of infrastructure, and the possibility of new types of prior context, as in the scenario described in Section 1.1. Our goal is to find a solution for key agreement in this scenario in a general ad-hoc network environment without making additional assumptions about the availability of any support infrastructure.

1.3 Password-based Authenticated Key Exchange

We can capture the existing shared context by choosing a fresh password and sharing it among those present in the room (e.g., by writing it on a blackboard). If this password is a sufficiently long random string, it can be used directly to set up a security association. In practice, people find it difficult to use long random strings. It is much more user-friendly if the password is a string that people recognize easily, such as a natural language phrase. But natural language phrases are *weak* as secrets because they are drawn from a rather limited set of possibilities; they are susceptible to *dictionary attacks* where an attacker records the encrypted traffic and then attempts trial decryptions with candidate passwords until he finds the correct one. Therefore, we need a protocol to derive a strong shared session key from the weak shared password.

Before we examine concrete protocols, we summarize desirable properties they should have.

- *Secrecy*: The basic secrecy requirement is that only those players that know the initial password should learn the resulting session key. All other observers must not be able to learn any information about the session key. *Perfect forward secrecy* requires that if an attacker who succeeds in compromising one of the players at a later time (and learns all permanent secret information known to that player) will still be unable to to discover session keys resulting from previous runs of the protocol.
- *Contributory key agreement*: If the resulting session key is composed of contributions by all players, the protocol is said to be be contributory [3]. The motivation for this requirement is that contributory protocols can be designed to assure a correctly behaving player that if his contribution is chosen randomly from a certain key space, even a coalition of all the other players will not be able to force the choice of the eventual session key to be limited to a smaller key space.
- *Tolerance to disruption attempts*: The strongest attacker can disrupt any protocol by jamming the radio channel or modifying communication among the legitimate participants. However, a slightly weaker attacker who can insert messages but cannot modify or delete messages sent by others is also of interest in this scenario. We will consider disruption tolerance against this type of disruption attacks only.

In the rest of the paper, we explore various alternatives for designing such a protocol. In Section 2 we describe a well-known two-party protocol for password authenticated key exchange and examine how it can be extended to multiple parties. In Section 3 we look at how password authentication can be added to multi-party Diffie-Hellman key exchange protocols. In particular, we present a fault-tolerant version of multi-party Diffie-Hellman key exchange protocol which can be of independent interest. In Section 4 we take a brief look at a related problem of how to elect a leader or agree on an ordering among ad-hoc networks nodes. In Section 5 we step back and discuss security issues in general in ad-hoc networks. Finally, in Section 6 we conclude with a summary.

We use the term “player” to refer to anyone who participates in a protocol, either as a legitimate participant, or as an attacker.

2 Generic Protocol for Password Authenticated Key Exchange

In a landmark paper [4], Bellovin and Merrit described a protocol called *encrypted key exchange* (EKE) using which two parties could derive a strong shared key starting from only a weak shared key. There has been significant subsequent work, especially on one variant of EKE based on Diffie-Hellman key exchange. But the basic form of the generic protocol remains the same. There are two parties A and B which share a weak secret P. A has a random key pair (E_A, D_A), for encryption and decryption respectively, in some suitable asymmetric key cryptosystem. The goal of the protocol is for A and B to mutually authenticate each other based on P, and to agree on a strong session key K, in such a way that an attacker watching the traffic will not be able to learn K or mount a dictionary attack on P. The basic protocol is described in Section 2.1 of [4] with an additional safeguard described in Section 2.5.3 of the same. During the protocol, A generates two random strings: $challenge_A$ and S_A. B generates three random strings: R, $challenge_B$ and S_B. A suitable symmetric key encryption scheme in which R can be used as a key is also assumed. The lengths of the other random strings should be specified in any concrete instantiation of this protocol. Finally, a public function $h(\cdot)$, and a public, one-way function $f(\cdot\,,\cdot)$ are also assumed. The protocol proceeds as follows:

(1) A $\rightarrow$ B: A, $P(E_A)$
(2) B $\rightarrow$ A: $P(E_A(R))$
(3) A $\rightarrow$ B: $R(challenge_A, S_A)$
(4) B $\rightarrow$ A: $R(h(challenge_A), challenge_B, S_B)$
(5) A $\rightarrow$ B: $R(h(challenge_B))$

In step 1 A sends E_A encrypted with the weak secret P. A also sends a label identifying itself. We assume that this message establishes a pair-wise communication channel between A and B. Therefore, no identification labels are included in the subsequent steps. B extracts E_A, generates R randomly, encrypts it with E_A, and returns it to A in step 2. A extracts R from this message, generates $challenge_A$ and S_A, encrypts them using R as key, and sends the result to B in step 3. B extracts $challenge_A$ from this message and computes $h(challenge_A)$. It generates its own random strings $challenge_B$ and S_B, encrypts them along with $h(challenge_A)$ using R as key and sends the result to A in step 4. A extracts the three quantities from this message and verifies that the first quantity is indeed $h(challenge_A)$. At this point, A can be convinced that B was able to extract $challenge_A$ from the message sent by A in step 3. This in turn implies that B knows P (otherwise B could not have correctly constructed the message in step 2, from which A extracted R used to encrypt the message step 3). A then computes $h(challenge_B)$, encrypts it using R as key, and returns the result to B in step 5. B decrypts this message, and verifies that the plaintext is indeed $h(challenge_B)$. B can now be certain that A was able to extract $challenge_B$ from message in step 4, which in turn implies that A knew P to be able to extract R correctly from message in step 2. At this point, each player will compute the session key as $K = f(S_A, S_B)$.

One obvious way to extend this protocol to the multi-party case is to elect a leader. The leader will broadcast the message in step 1. The rest of the messages will be point-to-point flows similar to the above protocol with A playing the role of the leader. At the end of the protocol run, each player shares a key with the leader. An additional round will be needed for the leader to pick a common session key and distribute it to the members of the group using the pairwise session keys he shares with them. The main drawback of this version is that the common session key is chosen unilaterally by the leader: the key agreement scheme is *non-contributory.*

We can slightly modify this generic two-party protocol so that it can be easily extended to a contributory multi-party protocol. The challenges $challenge_A$ and $challenge_B$ are used by A and B respectively to confirm that the other player does in fact know the password. Each player already generates a random quantity (*viz.* S_A and S_B), and we could use them instead of separate challenges. Since these quantities are used to generate K, it would be sufficient to confirm each other's knowledge of K. The modified protocol proceeds as follows:

(1) A $\rightarrow$ B: A, $P(E_A)$
(2) B $\rightarrow$ A: $P(E_A(R, S_B))$
(3) A $\rightarrow$ B: $R(S_A)$
(4) A $\rightarrow$ B: $K(S_A, H(S_A, S_B))$
(5) B $\rightarrow$ A: $K(S_B, H(S_A, S_B))$

The last two steps are used by the receiving party to confirm that the sending party knows K (and therefore P). In those steps $H(\cdot, \cdot)$ is a public hash function. Messages in steps 3 and 4 could be sent in the same flow. Extending this version to the multi-party case is straight forward. Let the set of players be $\{M_i,\ i = 1, \ldots, n\}$ and M_n be the leader. S_i is the random share contributed by M_i towards the eventual session key. M_n has a key pair (E, D) from a suitable asymmetric encryption scheme.

(1) $M_n \rightarrow$ ALL: M_n, $P(E)$
(2) $M_i \rightarrow M_n$: M_i, $P(E(R_i, S_i))$, $i = 1, \ldots n-1$
(3) $M_n \rightarrow M_i$: $R_i(\{S_j, j = 1, \ldots, n\})$, $i = 1, \ldots, n-1$
(4) $M_i \rightarrow M_n$: M_i, $K(S_i, H(S_1, S_2, \ldots, S_n))$, for some i

The session key is computed as $K = f(S_1, S_2, \ldots, S_n)$, where $f(\cdots)$ is an n-input one-way function. [2]

[2] A one-way hash function can serve as an n-input one-way function by concatenating the n inputs and using the result as the input to the one-way hash function. It is preferable to use one-way hash functions which hide all input bits. All commonly used one-way hash functions do have this property in practice.

The last step is used for key confirmation. It is enough for each player to confirm that one other player knows the same key. If step 3 was completed successfully, every player except M_n will have received key confirmation from one other player, namely M_n. It is sufficient if M_n randomly picks one of the other players and ask it to carry out step 4.

In the rest of this section, we summarize, without proofs, the extent to which the requirements listed in Section 1.3 are met. The multi-party protocol above, like the two-party version, provides perfect forward secrecy for all parties except the one who knows the decryption key D, unless the decryption key is also destroyed at the end of the protocol run. That is, an attacker who subverts M_n will be able to decipher the transcript of a past session. The key agreement is contributory since each player receives the contributions of the others relayed via M_n and then constructs K by combining these contributions with his own. The protocol is also tolerant of disruption attempts by anyone except M_n: if an attacker does not know P, it could attempt to send garbage in step 2. As a result legitimate players will agree on a session key which has a "contribution" from the attacker. But the attacker cannot determine this key. Note that making the protocol contributory has a price: M_n has to wait long enough in step 2 to gather the contributions from all players before it can start sending out the messages in step 3. In the non-contributory version, M_n can respond to each message in step 2 without delay.

This protocol has the following drawbacks. First, any quantity encrypted using P, such as E in step 1, should be random. This means that E cannot be a well-known long term public encryption key; instead it is preferable to use a fresh keypair generated for each run of the protocol. Generating a new keypair each time can be computationally expensive. Also, the parts or all of the encryption key E may have special properties (e.g., a product of large prime numbers, as in RSA) which will help an attacker attempting a dictionary attack on $P(E)$. Care must be taken to encrypt with P only the unpredictable parts of E. When using some asymmetric key cryptosystems, it may be possible to send the encryption key E unencrypted in step 1. This implies that after receiving the message in step 1, M_i $(i = 1, \ldots, n-1)$ do not know if E was sent by the legitimate leader M_n or an attacker. In particular, depending on the cryptosystem, an attacker may have chosen E in such a way that $E(R_i, S_i)$ is not random (even when R_i and S_i are random) leaving step 2 vulnerable to dictionary attacks. Thus, the message in step 1 may have to be accompanied or followed by an additional protocol that can convince M_i that E is a correctly formed encryption key for the chosen asymmetric key cryptosystem. These issues are discussed in detail in [4,5].

Next we look at password authenticated Diffie-Hellman key exchange protocols which avoid the difficulties with the generic protocol mentioned above.

3 Password authenticated Diffie-Hellman key exchange

3.1 Two-party version

Diffie-Hellman key exchange (DH) is the classic two-party key agreement protocol proposed in the paper that launched the era of modern cryptography [6]. In the basic DH protocol, two players A and B agree on a prime p and a generator g of the multiplicative group $\mathbb{Z}_p^*$ (i.e., the set {1, 2, ... p-1}). A and B randomly choose secrets S_A and S_B such that $1 \leq S_A, S_B \leq p-1$. (More generally, instead of $\mathbb{Z}_p^*$, any order q cyclic group can be used along with randomly chosen secrets drawn from $\mathbb{Z}_q$).

One version of password authenticated DH key exchange presented in the original paper [4] is as follows:

(1) A $\rightarrow$ B: A, $P(g^{S_A})$
(2) B $\rightarrow$ A: $P(g^{S_B}), K(C_b)$
(3) A $\rightarrow$ B: $K(C_a, C_b)$
(4) B $\rightarrow$ A: $K(C_a)$

In step 1, A computes g^{S_A}, encrypts it with P and sends it to B. B extracts g^{S_A} from this message, computes g^{S_B}, and computes the session key K as $K = (g^{S_A})^{S_B}$ (i.e., $K = g^{S_A S_B}$). B then chooses a random challenge C_b and encrypts it using K as the key, and encrypts g^{S_B} using P as the key and sends these two encryptions in step 2. A can now extract g^{S_B}, compute K as $K = (g^{S_B})^{S_A}$, decrypt the second part of the message and extract C_b. A then picks his own challenge C_a, encrypts both C_a and C_b using K as key and sends the result to B in step 3. B can decrypt this and check if the second part of the plaintext is indeed C_b. This would convince B that A was able to correctly decrypt the message in step 2, and therefore knew P. Then B encrypts C_a using K as key and sends the result to A in step 4. A can decrypt this to see if the plaintext is indeed C_a. This would convince A that B knew K. In order to correctly compute the same K as A, B must have known how to decrypt the message in step 1. This would convince A that B knew P.

3.2 Multi-party version

An unpublished protocol by Steiner et al [8], achieves authenticated DH group key distribution for a multi-party scenario that is slightly different from ours. They considered a group of n players who want to agree on a session key; each player shared a (different) password with an authentication server. However, we can use their protocol for our scenario as well. In our scenario, there are n

players $M_1, M_2, \ldots, M_n$ who all share a password P. Each player M_i generates a random quantity S_i, which is that player's share in the eventual session key. Our goal is that at the end of the protocol all players who knew P will end up with a shared session key $K = g^{S_1 S_2 \ldots S_n}$ where S_i is the random share contributed by M_i. The protocol is divided into three parts. In the first part (stages 1 and 2 in the description below), $M_1, \ldots, M_{n-1}$ construct an intermediate key $\pi = g^{S_1 S_2 \ldots S_{n-1}}$ in $n-1$ steps. All messages up to this point are sent in the clear. Note that M_n can compute the session key K already after this stage by raising π by S_n: $K = \pi^{S_n}$. In the second part (stages 3 and 4 below), each M_i $(i = 1, \ldots, n-1)$ engage in a separate exchange with M_n at the end of which all players have sufficient information to compute K. The first stage of this second part consists of messages encrypted using P as key. In the third part (stage 5) players perform key confirmation. $H(\cdots)$ in stage 5 is a public function used in the key confirmation message. Since all players share the same password P, it is enough if each player verifies that at least one other player has computed the same session key K.

(1) $M_i \rightarrow M_{i+1}$: $g^{S_1 S_2 \ldots S_i}$, $i = 1, \ldots, n-2$, in sequence
(2) $M_{n-1} \rightarrow$ ALL: $\pi = g^{S_1 S_2 \ldots S_{n-1}}$, broadcast
(3) $M_i \rightarrow M_n$: $P(c_i)$, $i = 1, \ldots, n-1$, in parallel, where $c_i = \pi^{\hat{S}_i / S_i}$ and $\hat{S}_i$ is a blinding factor that is randomly chosen by M_i.
(4) $M_n \rightarrow M_i$: $(c_i)^{S_n}$, $i = 1, \ldots, n-1$, in parallel
(5) $M_i \rightarrow$ ALL: M_i, $K(M_i, H(M_1, M_2, \ldots, M_n))$, for some i, broadcast

Stage 1 contains $n-2$ steps. In the first step, player M_1 computes g^{S_1} and sends it to M_2. In step i, player M_i raises the quantity received in the previous step by S_i and sends it to M_{i+1}. At the end of stage 1, M_{n-1} has received $g^{S_1 S_2 \ldots S_{n-2}}$ which it can raise by S_{n-1} to obtain $\pi = g^{S_1 S_2 \ldots S_{n-1}}$. In stage 2, M_{n-1} broadcasts π to everyone. Now, each M_i $(i = 1, \ldots, n-1)$ removes its contribution from π (i.e, M_i raises π by $1/S_i$) but inserts a randomly chosen blinding factor (i.e., M_i raises π^{1/S_i} by the blinding factor $\hat{S}_i$). The resulting quantity, c_i, is encrypted using P as key. In stage 3, each M_i $(i = 1, \ldots, n-1)$ will, in parallel, send the encryption to M_n. M_n will decrypt the received messages to extract c_i. It then raises each c_i by S_n and in stage 4 returns the result, in parallel, to each M_i. The players can at this point compute the session key $K = g^{S_1 S_2 \ldots S_{n-1} S_n}$ as follows. M_n raises π by S_n: $K = \pi^{S_n}$. Each M_i $(i = 1, \ldots, n-1)$ unblinds the quantity received from M_n and re-inserts its original contribution to construct the session key: $K = (c_i^{S_n})^{S_i/\hat{S}_i} = \pi^{S_n}$. Finally, in stage 5 some (at least two different) players broadcast a key confirmation message that allows each player to verify that at least one other player has decided on the same key K. The quantity $H(M_1, M_2, \ldots, M_n)$ helps to verify that all members have the same view about the composition of the group. In scenarios where labels of members are meaningless, this can be omitted.

Blinding with $\hat{S}_i$ is necessary for two reasons. First, without the blinding, the quantity encrypted with P by M_{n-1} to be sent in stage 3 is the same as what it received in stage 1. A second, less compelling, reason is that an active attacker could feed $g^{S_1 S_2 \ldots S_i}$ (which the attacker can observe during stage 1) to M_i in stage 2 instead of the broadcast message π. If an unsuspecting M_i uses this quantity to construct its message for stage 3, the resulting message is the same as the message that M_i received in stage 1. Thus, to thwart dictionary attacks on a message in stage 3, it has to be blinded.

This protocol provides perfect forward secrecy to all players. It is also partially resilient to disruptions: The leader, M_n, can disrupt the protocol completely. Any other other player may attempt to disrupt the protocol by sending out a random quantity in stage 1 unrelated to the one received by it. If M_i did this, none of $M_1, \ldots, M_{i-1}$ will be able to compute the session key. However, $M_{i+1}, \ldots, M_n$ can compute a common session key.

3.3 Fault tolerant Diffie-Hellman key exchange on a d-cube

Becker and Wille [9] showed how multi-party Diffie-Hellman key exchange can be done efficiently, in terms of the number of communication rounds, by arranging the players in a d dimensional hypercube. A d dimensional hypercube, also called a d-cube, is a graph in the form of a 'cube' with 2^d vertices, each of which is connected to d of the other vertices. Each vertex is assigned a unique d-bit address drawn from the set $\mathbb{Z}_n$ such that the addresses of two vertices that are connected by an edge along the j^{th} dimension of the cube differ only in their j^{th} bit.

In this section, we describe the basic Becker and Wille approach and show how it can be extended to handle failures.

3.3.1 Diffie-Hellman exchange on a d-cube

The basic idea proposed by Becker and Wille [9] is illustrated in Figure 1 with four players A, B, C, and D who want to agree on a key. They have two-bit addresses 00, 01, 10, and 11 respectively.

Suppose they use S_A, S_B, S_C, and S_D respectively as their random shares for 2-party DH. In the first round, A and B run 2-party DH and compute $S_{AB} = g^{S_A S_B}$ while C and D engage in a parallel run of 2-party DH resulting in $S_{CD} = g^{S_C S_D}$. In the next round, A runs a 2-party DH with C while B runs a 2-party DH with D. Each player uses the key computed in round 1 as their

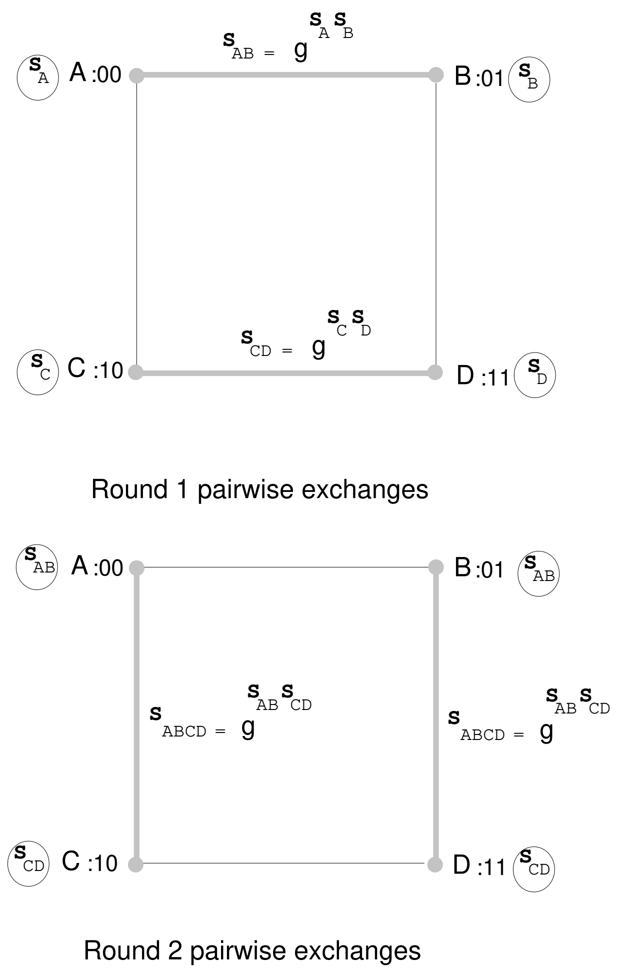

Fig. 1. Four-party DH key exchange. Gray edges connect players that perform two-party DH key exchange. The quantities in circles indicates the secret known by a player at the beginning of the round. The quantities near the edges indicates that shared secret computed in the round.

secret share for round 2. [3] At the end of the round, all four players will have the same key $S_{ABCD} = g^{(S_{AB} S_{CD})}$. If we imagine the players as vertices of a square, the exchanges in each round consisted of edges in the square along the same dimension.

Now, suppose there are $n = 2^d$ players. Each player is assigned to a vertex in the hypercube. Thus each player has a unique d-bit address drawn from the set $\mathbb{Z}_n$. We can therefore imagine that the 2^d players are arranged in a d-dimensional hypercube. The protocol proceeds through d rounds, $1, \ldots, d$. Consider the player with address I. In the j^{th} round, this player performs a two-party DH with the node whose address is $I \oplus 2^{j-1}$. In other words, in the

[3] As pointed out in [9], in the general DH setting where an order q subgroup G of $\mathbb{Z}_p^*$ is used, $g^{S_A S_B}$ has to be mapped to $\mathbb{Z}_q$ by means of a suitable function before it can be used in the next round two-party DH exchange.

j^{th} round, neighbors along the j^{th} dimension of the hypercube participate in a two-party DH protocol run. After d such rounds all players will have the same key.

Induction on the number of players n is the easiest way to see the Becker and Wille protocol works. The basic case of four players that agree on a shared key in two rounds is illustrated in Figure 1. Next, suppose that we know how to do multi-party DH key agreement when $n = 2^{k-1}$, where k is an integer greater than 2, and we wish to extend the protocol to $n = 2^k = 2^{k-1} + 2^{k-1}$. In this case we have two groups of size 2^{k-1}. By our inductive assumption the members of each group can do multi-party DH between themselves in $k - 1$ rounds, independently of the other group. In the end of those $k-1$ rounds each group will have a shared key. For all 2^k members to agree on a single shared key in round k, each member of one group has to perform two-party exchange with a player who knows the key of the other group. The fastest way to do this is to pair each member of one group with a different member of the other and then perform 2^{k-1} two-party key exchanges in parallel, independently of the other pairs. The time needed will be the same as that of one two-party key exchange. The d-cube has just the right connectivity to allow parallel key exchange between the two groups.

Notice that in round 1 there are 2^{d-1} pairs of groups, each group consisting of a single member, that exchange keys independently of the other pairs. In round 2 there are 2^{d-2} pairs of groups, each group consisting of 2 members, that exchange keys independently of the other pairs. In round k, there are 2^{d-k} pairs of groups, each group of size 2^{k-1}, doing key exchange in parallel.

3.3.2 Adding password authentication

It is obvious how authentication based on a single shared password can be incorporated into this "cube protocol": Password authenticated n-party DH can be achieved by simply using the four move two-party password authenticated DH protocol from Section 3.1 for pair-wise exchanges in each round of the d-cube protocol. We are left with two questions: what if the number of players n is not a power of 2, and what if a two-party key exchange fails because one of the parties is faulty or because one of the parties is an attacker that tries to prevent honest players from reaching key agreement?

For the case where n is not a power of 2, Becker and Wille present a protocol which they call '2^d *octopus*' to optimize the number of rounds. If $2^d < n < 2^{d+1}$, then the first 2^d players play the role of central controllers. The remaining $n-2^d$ players are distributed among the central controllers as their wards (i.e., at most one per controller). In the first stage, the controllers execute two-party DH with their ward. Then, the controllers engage in a d round cube protocol run using the information gathered in the first stage. In the third stage, the key derived in the second stage is distributed to the wards. See [9] for details.

In our setting, a player may discover during any round that the key exchange with its chosen partner failed. If this happens, this player needs to find an alternate partner to do the exchange. It is not clear how the Becker and Wille optimization can be adapted to take exchange failures into account.

3.3.3 Dealing with faults

If a node finds that its chosen partner in a given round is faulty, we need an algorithm by which it can select other potential partners until a non-faulty one is found. Consider any such partner selection algorithm P. Since the number of potential partners in round k is bounded by 2^{k-1}, P can have at most 2^{k-1} sub-rounds. Let P(k, i, I) denote the the node address of the partner suggested to node I by P in round k, sub-round i. We can see two requirements on P(k, i, I):

(1) P must not match two nodes to the same partner in a given sub-round. Therefore, for a given k and i, P(k, i, I) must be unique for each I.
(2) P must not select the same partner twice for a given node. Therefore, for a given k and I, P(k, i, I) must be unique for each i.

One can think of several algorithms that satisfy these requirements. In Figure 2 we present a distributed algorithm using which a player can find a suitable partner (if such a partner exists) for the two-party exchange in the current round. That is, our algorithm works both when n is not a power of 2 as well as when attackers attempt to participate in the protocol. The algorithm is best illustrated by an example. Consider Figure 3. Players have three bit addresses and are labeled A-H. Their contributions to the key are represented by the corresponding lower-case letters. Labels next to the arrows indicate the players that have contributed, directly or indirectly, to the key resulting from the exchange indicated by the arrows. Suppose the player G (with address 110) is unsuitable (unavailable or does not know the password). In round 1, player H (111) will start the partner finding procedure with 110 as the candidate address and 000 as the mask. The exchange attempt with G fails and the mask is already 000. So H does nothing in this round. In round 2, E (100) will start with 110 as candidate address and 001 as mask. The first recursive call will try 110 as candidate address and 000 as mask and will fail. The second recursive call will try 111 as candidate address and 000 as mask and will succeed. Similarly, in round 3, node C (010) starts partner finding with 110 as candidate. Figure 4 shows the sequence of partner choices examined by node C.

```
Node algorithm for each round
procedure do_round(round_number)
  mask := 00...01 // Initialize a mask of d bits
                 // by setting the least significant bit
  mask := mask << round_number-1 // left shift mask by
                              // 'round_number-1' bit positions
  partner := self_address ⊕ mask // '⊕' indicates
                                 //the XOR operation
  new_mask := mask >> 1 //right shift mask by one bit position
  two_party_exchange(partner, new_mask)
end


Recursive sub-algorithm for finding a partner and performing two-
party exchange
procedure two_party_exchange(candidate, mask)
  if (mask <= 0)
    // We have reached a leaf node
    attempt running the two-party DH key exchange
    return success or failure
  endif

  // Otherwise, we are still at a non-leaf;
  // try the left side first

  new_mask := mask >> 1 //right shift mask by one bit position
  result := two_party_exchange(candidate, new_mask)
  if (result = success) return success;

  // left side invocation did not succeed; try the right side
  new_candidate := candidate ⊕ mask
  return two_party_exchange(new_candidate, new_mask)
end
```

Fig. 2. Distributed algorithm for finding partners in fault tolerant n party key exchange.

3.3.4 Analysis

This algorithm suggests the "closer" partners before the more distant ones, in terms of the euclidean distance between the node addresses. This can be a useful property if the underlying network is a hypercube.

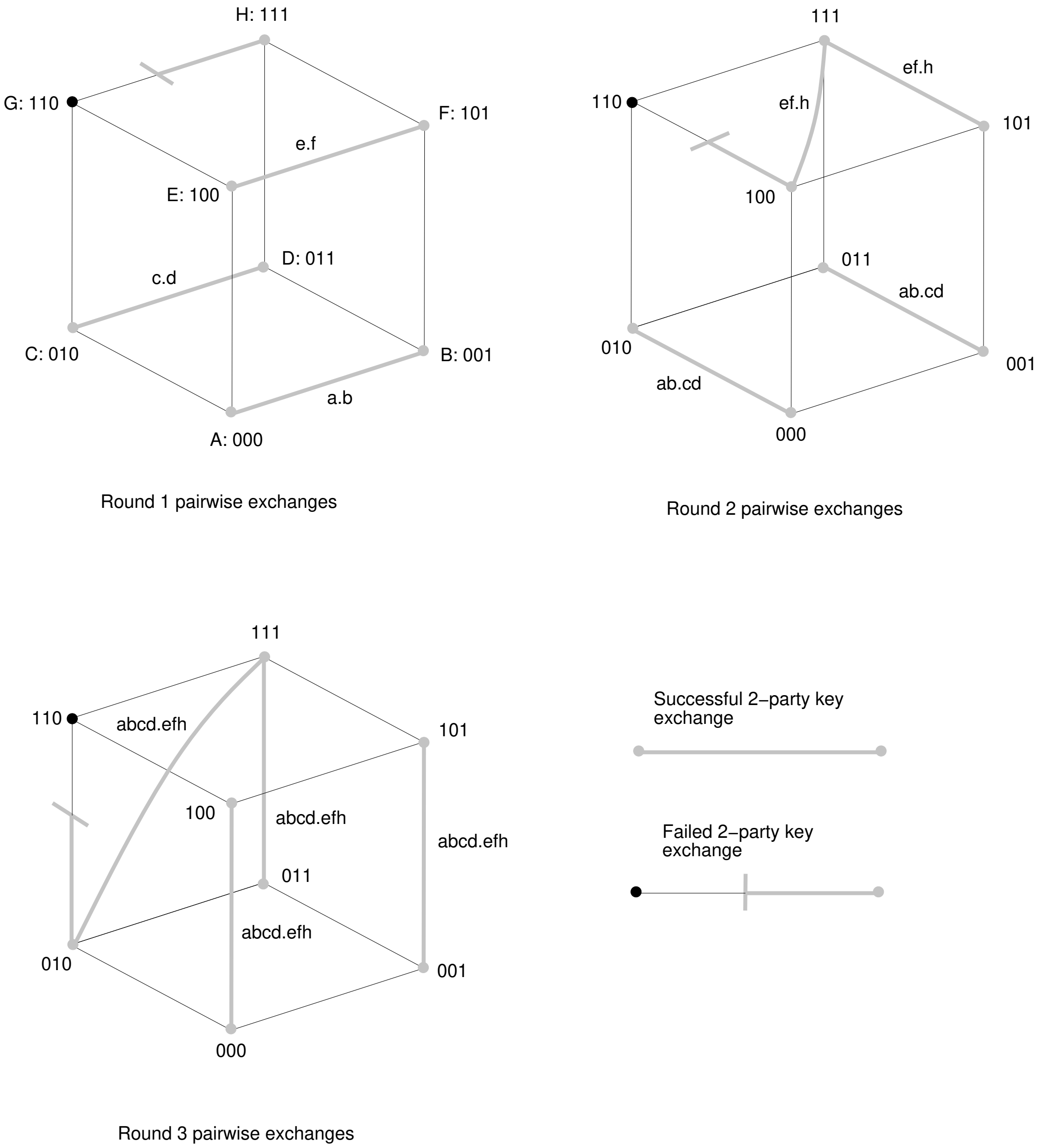

Fig. 3. Fault tolerant 8-party key exchange. In each round a different neighbor of node G will discover that G doesn't know the password P. When this happens, the neighbor will use the algorithm on Figure 1 to pick another partner.

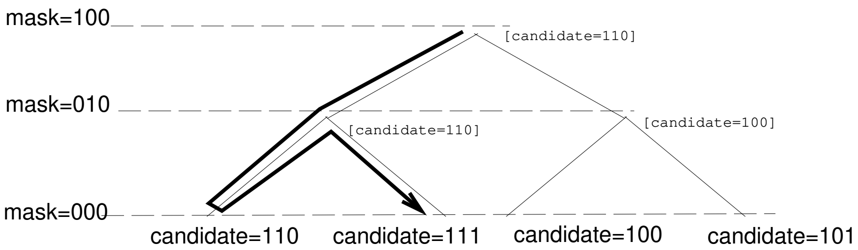

Fig. 4. Selection of partner in Round 3 by node C (010) in Figure 3.

We can compare the 'no-faults' case of our protocol with the 2^d *octopus* protocol. We have described the protocol in terms of rounds. Each round may consist of several sub-rounds when two-party key exchange with the default partners cannot be done. The operation of a player during a given sub-round consists of two phases: first, it computes all outgoing messages and sends them; then it reads all waiting messages and changes state accordingly. Intuitively, a sub-round lasts long enough to perform one 2-party Diffie-Hellman key exchange. Our measure of communication complexity is the total number of sub-rounds.

When the number of players n is not a power of 2, $2^d < n < 2^{d+1}$, the 2^d *octopus* protocol will take $d+2$ rounds in total, each round consisting of only one sub-round. In our protocol the number of rounds is limited to $d+1$, but the number of sub-rounds in a round can be as big as 2^d, when $n = 2^d + 1$. However, by labeling the players differently the worst-case complexity of our protocol can also be brought down to *O(d)* rounds. Starting from 0, keep assigning even numbered node addresses to the "missing" $2^{d+1} - n$ players until they are all accounted for. The rest of the addresses are assigned in sequence to the remaining actual participants. In each round, any player whose default partner is a missing player is guaranteed to find an active partner in the next sub-round (since it will be a player with an even numbered node address). For example, when there are 10 players, we assign addresses 0, 2, 4, 6, 8, and 10 to the "missing" players, while the actual players receive the addresses: 1, 3, 5, 7, 9, 11, 12, 13, 14, 15. With this modification, the worst case complexity is $2(d+1)$ sub-rounds.

Let us now look at the number of communication rounds required for fault-tolerant multi-party DH key exchange on a d-cube when using any partner selection algorithm P which satisfies the requirements listed in Section 3.3.3.

Suppose we need to run d rounds: i.e., the total number of players is not more than 2^d. Assume that m of these players are faulty (including both "missing" players as well as really faulty players). Let $2^k \leq m < 2^{k+1}$ for some $0 \leq k < d$.

Consider the following pathological layout of faulty players. $2^k - 1$ of them are located in a single k-cube C1; the remaining $m - 2^k - 1$ are located in the k-cube C2 which will participate in a round $k+1$ exchange with C1.

The number of sub-rounds required in rounds $k+2$ to d, where $k < d-1$, is at most $m+1$ per round. This is because in each of those rounds, there is always one sub-cube with m faulty players. The same non-faulty player may select using P each of the m faulty players in sequence before being able to complete its round exchange, resulting in $m+1$ rounds. From requirement (2) in Section 3.3.3 it follows that this is the maximum number of sub-rounds required for that player. Since there is no other sub-cube with more faults, $m+1$ is the maximum number of sub-rounds required.

In round $k+1$, the number of faulty players in cube C1 is $2^k - 1$. Some player in cube C2 may try all the faulty players in C1 before finding the only non-faulty player. Thus the maximum number of sub-rounds required for round $k+1$ is 2^k. In round j, $1 \leq j < k$, the only non-faulty player will have to try all possible partners before giving up, at the cost of 2^{j-1} sub-rounds. The total for the first $k+1$ rounds is therefore $\sum_{j=0}^{k} 2^j = 2^{k+1} - 1$.

Thus the total number communication rounds required to complete the exchange in this scenario is $2^{k+1} - 1 + (d-k-2)(m-1)$, when $0 \leq k < d-1$ (when $k = d-1$, the second term above does not occur). We can show that this is the upper bound for the number of communication rounds with m faults, regardless of the layout of the faults. Suppose the faults were distributed differently among the 2^{r-1} sub-cubes, where $k+2 \leq r \leq d$. The number of faults in any of these sub-cubes will be strictly less than m. Hence the number of sub-rounds required in each round from $k+2$ to r will be less than $m+1$. Our pathological example already incurs the maximum possible number of sub-rounds in the worst case during rounds 1 through $k+1$.

When there are attackers, our protocol retains the nice property that as long as attackers cannot remove or modify messages sent by honest players (i.e., attackers can only *send* spurious messages), there is no denial of service attack. Perfect forward secrecy and the contributory nature of two-party DH are retained as well.

3.3.5 Implementation considerations

The partner selection algorithm in Section 3.3.2 is described in terms of rounds. Synchronous rounds could be implemented if all nodes have loosely synchronized clocks. But it is also possible to allow each node to proceed independently. When a node A initiates a pair-wise exchange with its chosen partner B, A can indicate the round number i. If B has not yet reached round i, it will reply with a 'try later' status message, causing A to block on this round. This might cause a cascade of blocked nodes. When B replies, or when A times out and finds some other partner for round i, the cascade will be unblocked. On the other hand, if B has already completed round i, presumably with some other partner than A, B will reply correctly if g^{S_A} is the same as the quantity received by B in the previously completed round i exchange. Otherwise B will signal an error. This way different players could be at different rounds at any moment of time.

The algorithm is based on the assumption that *all* pair-wise exchange attempt with a faulty player will fail. This is the case if the faulty player is an attacker. In practice, we also need to consider resilience in the face of benign failures. For simplicity, assume that rounds proceed in a synchronous fashion. If a node participates successfully in the first i rounds but crashes at the end of a round i, the rest of the nodes can continue the protocol successfully. When rounds are independent, crashed nodes that recover later can also successfully complete the protocol.

The most general benign failure model is to allow node crashes in the middle of a round. A small modification to the algorithm in Section 3.3.2 makes it resilient to this type of faults. Let the subset of nodes having the same $d - i$ most significant bits in their addresses be defined as the set of *round i peer set* (e.g., A, B, C, and D are round 1 peers in Figure 3). Suppose that during round i, all members of a round i peer set crash in the middle. As a result, not all members of the partner peer set may have finished their round i exchanges successfully. The change to the algorithm is simple: when a player cannot find any suitable partner in round i, it should check with all members of its round i peer set to see if they too failed to find a partner. If any member of the round i peer set has succeeded, it can simply provide the result of the round i exchange to the others in the set.

4 Deciding on leaders and ordering

All of the protocols above required either electing a leader or choosing an ordering among the players or both. In most well-designed protocols, being a leader or occupying a particular position in the ordering does not appear to bestow any clear advantage in influencing the choice of the session key. But the possibility cannot be overruled: for example, in the password-based multi-party DH key exchange protocol (Section 3.2), the leader M_n has a greater say in the final session key than the other players (for example, M_n can try several values of S_n before finding one that leads to a particular type of K). It is not clear whether this can be translated into any tangible advantage.

In some ad-hoc networks there may already be a natural leader or ordering. However, in any of the protocols above we did not make any particular assumption about who can be the leader or what should the ordering be. Therefore, when there is no *a priori* leader or ordering, a natural question is whether we could devise a leader election or order-finding process that cannot be influenced by any one party. Electing a leader is a special case of determining an order. Therefore we will consider the latter.

The general approach to randomized order finding is simple: each player commits to a random number without revealing what the number is. When all commitments have been received and acknowledged by all players, the random numbers are revealed. A public function is then applied to the input random numbers to compute a reference value. This computation can be carried out independently by each player. The ordering is then determined by sorting the numbers according to their distance from the reference value.

5 Security issues in ad-hoc networks

We have examined a problem that arises in a specific ad-hoc network scenario. Nevertheless, we believe that this scenario captures one of the primary distinguishing features of ad-hoc networks from the point of view of security: all security protocols are based on some form of prior context among the players. What is different about ad-hoc networks is that we encounter *new* types of prior context that are typically not found in other networking scenarios. Stajano and Anderson [10] discuss some security issues specific to ad-hoc networking scenarios. They identify denial of service, goals of authentication, and problems of naming as issues in which ad-hoc networks differ from conventional systems.

There are security problems that do not yet have satisfactory solutions even in standard contexts: How can security services be visualized so that a novice user can easily configure and use them? How can trust management policies be expressed and enforced? How can security policies at various levels (organizational, personal etc.) be combined? Possibly, these problems need to be solved well before secure ad-hoc networking can be a reality. However, the initial focus of security research in ad-hoc network ought to be on those problems that are well understood in standard network scenarios but are difficult to solve, or even to characterize, in ad-hoc network scenarios. Key agreement, which is a fundamental building block for various security services, is such a problem.

Sometimes ad-hoc networks are defined as "infrastructure-less." However, this is likely to be an extreme case only. Networks with partial infrastructure will possibly be common. Availability of some infrastructure will make it easier to provide routing, security and other services. As a first step, the types of partial infrastructure that are likely to be common in ad-hoc networks should be identified and characterized.

6 Conclusions

We have introduced a new key agreement scenario and examined various solutions to the key agreement problem in this scenario. We showed how the generic two-party encrypted key exchange can be extended to the multi-party case. In addition, we have introduced a variation of the generic protocol extending which we achieve a contributory multi-party key agreement. We also described previously known protocols for password authenticated multi-party Diffie-Hellman key exchange, and presented improvements to one of them (the cube protocol) to make it resilient to malicious and benign failures.

Password-based key agreement meets the basic requirement we set in Section 1.2: it solves the problem without requiring additional infrastructure or physically secure communication channels. Password-based key agreement is also applicable in other scenarios. For example, consider a group of people who happened to meet one another in person for the first time and would like to go back home later to set up a secure channel among themselves. They could employ the same solutions without any modifications. A second application may be in personal networking or home networking where many devices owned by the same user need to share a key. For example, the Bluetooth industrial consortium is developing specifications for ad-hoc networking using short-range radio communication [11]. Key management in Bluetooth is bootstrapped from a PIN code which typically consists of four decimal digits. The Bluetooth specifications (version 1.0 A, Section 14.2.1) suggest that two party Diffie-Hellman key agreement can be utilized when keys derived from a short PIN are not sufficient. When multiple nodes are involved, as in a home Bluetooth network, our protocols may be applicable.

Several lines of future work are possible. Formal security analyses of the protocols presented here is a necessary missing step. A number password-based key agreement protocols have been proposed recently [12–14]. These protocols improve on the original EKE protocol. For example, Open Key Exchange protocol by Stefan Lucks [12] avoids the need to encrypt public keys. It will be useful to see if these protocols can be extended to the multi-party scenario considered here.

We have assumed that the composition of the group does not change during the session. If groups are dynamic, the session key needs to updated when the composition of the group changes. Finally, as we have described in Section 3.3.5, implementation of multi-party key agreement will need to address the issues of synchronization and resilience in face of benign faults.

7 Acknowledgments

We would like to thank Hannu Flinck, Jan-Erik Ekberg, Pekka Lahtinen, Valtteri Niemi, and Michael Steiner for their feedback on previous versions of this paper. As usual, it is a pleasure to thank Michael Steiner for several interesting discussion sessions on the subject.

References

[1] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). IETF Network working group request for comments, November 1998. RFC 2409.

[2] Roy Want, Andy Hopper, Veronica Falcäo, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[3] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *5th ACM Conference on Computer and Communications Security* [15], pages 17–26.

[4] Steven M. Bellovin and Michael Merrit. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1992.

[5] Sarvar Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 236–247, Oakland, CA, May 1997. IEEE Computer Society Press.

[6] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[7] Michael Steiner, Gene Tsudik, and Michael Waidner. Refinement and extension of *encrypted key exchange. ACM Operating Systems Review*, 29(3):22–30, July 1995.

[8] Michael Steiner, Gene Tsudik, and Michael Waidner. Private communication. Unpublished work described in slides of presentation made at the Third ACM CCS conference, March 1996.

[9] Klaus Becker and Uta Wille. Communication complexity of group key distribution. In *5th ACM Conference on Computer and Communications Security* [15], pages 1–6.

[10] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, April 1999. Available from http://www.cl.cam.ac.uk/ fms27/duckling/.

[11] Bluetooth. Specification of the Bluetooth system. Bluetooth Consortium, 1999.

[12] Stefan Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Security Protocol Workshop'97*, Ecole Normale Suprieure, Paris, April 1997.

[13] David P. Jablon. Extended password key exchange protocols immune to dictionary attack. In *Proceedings of the WETICE'97 Workshop on Enterprise Security*, Cambridge, MA, USA, June 1997.

[14] Thomas Wu. The secure remote password protocol. In *Symposium on Network and Distributed Systems Security (NDSS '98)*, pages 97–111, San Diego, California, March 1998. Internet Society.

[15] *5th ACM Conference on Computer and Communications Security*, San Francisco, California, November 1998. ACM Press.