Private Channels in Wireless Local Networks

Maurizio Adriano Strangio

Department of Information, Systems and Production, University of Rome "Tor Vergata", Rome, ITALY

Summary
This paper presents a modified version (DHBT-E) of an existing two-party key agreement protocol (DH-BT) used for efficiently establishing secure communication sessions in local wireless networks. A formal security analysis in the model of Bellare and Rogaway is provided to justify the relevant security properties and thus to enforce trustworthiness.

The basic intuition is that key agreement in open-air-networks should provide forward secrecy, since eavesdropping is essentially for free and hence may give the adversary a small but significant advantage. For this reason, the protocol uses basic Diffie-Hellmann key exchange.

Entity authentication is based on the shared string model, with the key formed by two cryptographic component keys (one key being eventually a low-entropy human memorable string), thus guaranteeing a stronger (two-factor) level of security.

Key words:
Wireless networks, Diffie-Hellman key exchange, Symmetric key authentication, Key agreement protocol

1. Introduction

This paper describes a modified version (DHBT-E) of the DH-BT implicitly authenticated key agreement protocol presented in [1]. The protocol is targeted for resource-constrained devices communicating via local wireless network technologies (e.g. Wi-Fi, IrDA, Bluetooth). We expand the informal arguments given to justify the security of the protocol in [1] and provide a more detailed analysis to prove the relevant security properties and thus to enforce trustworthiness. In particular, a formal security proof is developed in the formal model of distributed computing originally proposed by Bellare and Rogaway [2].

Our main thesis is that key agreement protocols in open-air-networks should provide forward secrecy since eavesdropping is essentially for free (with low cost hardware and open source software tools). For this reason, the protocol is based on basic Diffie-Hellman key exchange since forward secrecy is difficult to achieve in other ways.

The computational complexity of the protocol is close to optimum since only two exponentiations are required (only one, if pre-computation is possible). There is also a flexible choice of the underlying group that allows efficient storage and implementations (e.g. elliptic curve groups). It is hard to come up with a more efficient protocol under the specified security properties.

The protocol operates with two symmetric key-based authentication modes. The shared key is composed of a primary cryptographic key embedded in the code (a midlet that resides on the device) and a secondary key provided by the user.

In *manual mode*, the user must type in the secondary key (typically a human-memorable string such as a PIN or password) to complete a protocol run.

In *transparent mode*, the secondary key is provided by the user via a portable secure token (e.g. secure digital card or USB device).

2. Related Work

A protocol proposed by Maher [3] uses a hash function to authenticate Diffie-Hellman exponents exchanged by two principals. Both users input a key K into their devices and a one-way hash function h is used to obtain h(K); this value is truncated to the desired length by taking the most significant bits. However, this technique can be ineffective since it requires at least 48 bits of the hash to offer a reasonable amount of security and requires the devices to possess either a display or keypad. This protocol is specifically designed for Personal Area Networks (PANs).

Garefalakis and Mitchell [4] describe another solution for PANs based on Diffie-Hellman key exchange. The resulting protocol requires a device acting as a *Private Key Generator* (PKG) which is responsible, among other things, of issuing and distributing shared random data to

Manuscript received August 5, 2006.
Manuscript revised August 25, 2006.

every device forming the network. Again, users must manually enter data delivered by the PKG into their devices to authenticate the Diffie-Hellman exponents. This protocol also suffers from the drawbacks that were pointed out for the preceding one.

A third protocol based on string comparison has been recently presented by Cagalj *et al.* [5]. It is also based on Diffie-Hellman key exchange, authenticated using an ideal commitment scheme. The resulting protocol (DH-SC) has an increased communication complexity (in terms of the number of rounds and number of bits exchanged).

An advantage of the three aforementioned (manual authentication) protocols is that if the human assisted authentication phase fails the principals need not compute the Diffie-Hellman secret.

With protocol DHBT-E we avoid the problems pointed out above since a (secondary) key (held by the users) is required to compute the session key, at the expense of one more exponentiation.

3. The Key Agreement Protocol

3.1 Preliminaries

Given two strings s_1, s_2 the construct $s_1 | s_2$ denotes string concatenation. Sampling of the random variable x_i (according to the probability distribution D_i) over the set X_i is denoted by $x_i \xleftarrow{D_i} X_i$ (or by $x_i \xleftarrow{R} X_i$ if D_i is the uniform distribution). If X_i is neither an algorithm nor a set, $x_i \longleftarrow X_i$ represents a simple assignment statement. We denote by $\Pr[E_1; E_2; ...: A(x_1, x_2, ...; r) = y]$ the probability that event $A(x_1, x_2, ...; r) = y$ occurs in the context of an experiment where the sequence of events $E_1; E_2; ...$ have occurred first.

Let GD be an algorithm which on input 1^k outputs a description of a group G of prime order q (with $|q| = k$ bits) and a generator $g \in G$.

The security of many cryptosystems is based on the assumption that the computation of discrete logarithms in the group G is infeasible; this is known as the *Discrete Logarithm Problem* (DLP). More formally:

Assumption 1 [DLP] Algorithm GD satisfies the DL assumption if for all PPT algorithms A the probability of computing g^x for random x is negligible:

$$\Pr\begin{bmatrix} (G,q,g) \longleftarrow GD(1^k) \\ x \xleftarrow{R} Z_q^* \\ X \longleftarrow g^x \\ A(G,q,g,X) = x \end{bmatrix} < \varepsilon$$

where the probability is taken over the coin tosses of A (and random choices of x).

In a concrete security analysis we drop the dependence on k and consider the maximum probability over all adversaries A running in time bounded by t, in this case we say that GD (or simply G) is (t, ε)-*secure* with respect to the DLP.

Another assumption we shall make use of is the *Computational Diffie-Hellmann Problem* (CDHP). A formal definition is set out below.

Assumption 2 [CDHP] Algorithm GD satisfies the CDH assumption if for all PPT algorithms A the probability of computing g^{xy} given g^x, g^y is negligible:

$$\Pr\begin{bmatrix} (G,q,g) \longleftarrow GD(1^k) \\ x \xleftarrow{R} Z_q^*; y \xleftarrow{R} Z_q^* \\ X \longleftarrow g^x; Y \xleftarrow{R} g^y \\ A(G,q,g,X,Y) = g^{xy} \end{bmatrix} < \varepsilon$$

In a concrete security analysis we drop the dependence on k and consider the maximum probability over all adversaries A running in time bounded by t, in this case we say that GD (or simply G) is (t, ε)-*secure* with respect to the CDHP.

3.2 The protocol specification

Prior to execution of the protocol, two principals A, B (say, with respective identities ID_A, ID_B) agree upon the long-term shared key $K_{AB} = K_{BA} = \langle K_{1,AB}, K_{2,AB} \rangle$. The main protocol actions are summarised below (refer to Fig. 1 for the details):

(1) A, the initiator, selects a random number a in Z_q^* computes g^a and sends it to B;
(2) B, the responder, selects a random b in Z_q^* computes g^b and sends it to A;
(3) A and B compute their respective session keys sk_A, sk_B using the hash function H; if both hold the same shared key $K_{AB} (= K_{BA})$ then $sk_A = sk_B$.

$$A, B : K_{AB} = K_{BA}$$

$$\begin{aligned}
&A : a \xleftarrow{R} Z_q^* \\
&\quad X \longleftarrow g^a \\
&A \rightarrow B : X \\
&B : b \xleftarrow{R} Z_q^* \\
&\quad Y \longleftarrow g^b \\
&B \rightarrow A : Y \\
&A : sk_A \longleftarrow H(ID_A, ID_B, X, Y, Y^a, K_{AB}) \\
&B : sk_B \longleftarrow H(ID_A, ID_B, X, Y, X^b, K_{BA})
\end{aligned}$$

Fig. 1 Protocol DHBT-E.

The fundamental properties to be upheld by an implicitly authenticated key exchange protocol are the privacy and the authenticity of the resulting session key. To this end, it is required that the key and the exact identities of both principals must be inextricably linked to avoid one of A or B being fooled into associating the key with the wrong principal.
For protocol DHBT-E, only principals with knowledge of both $K_{AB}(= K_{BA})$ will possibly compute the exact same session key. The secrecy of the session key formed in an execution of the protocol rests upon the CDHP assumption described in Section 3.1.

Key confirmation is implicit, i.e. it is achieved by subsequent use of the session key by the calling applications. Explicit key confirmation can be added to the protocol (at the expense of increased communication complexity) by using the compilers of [6,7].

The devices are initialised with a midlet having the secret key $K_{1,AB}$ securely embedded in the code uploaded from a trusted computer via some authenticated link (e.g. IrDA, USB cable). Thereafter the protocol operates with essentially two authentication modes based on the secondary key $K_{2,AB}$:

manual: key $K_{2,AB}$ is a low entropy string (perhaps just a password or Personal Identification Number - PIN) which is manually typed into the devices by both parties to complete a protocol run. When the devices return into network range the users are re-prompted for the key if the preceding authentication period has expired or if the connection is lost due to the devices moving out of network range;

transparent: key $K_{2,AB}$ is read from a cryptographic token that is inserted into the device prior to a protocol run (the device is protected from unauthorized access, for example, with a pass-phrase or biometric data).

Notice that in both modes no information on K_{AB} is ever sent across the network.

4. Formal security model

Informally, a secure key agreement protocol should not allow a resource-bound adversary to manipulate the message flows in any (polynomial) number of protocol executions between honest parties, in such a way that information is leaked on the session key (or one or more protocol goals are subverted).
We try to capture this notion in the formal model of [2] in the symmetric key setting. A two-party key agreement protocol is defined as a pair $\Sigma = (\mathrm{K}G, \Pi)$ of poly-time computable functions where $\mathrm{K}G$ specifies how to compute the long-term private key and Π specifies the protocol actions and message formats. There is a finite number q_n of principals generically denoted by P_i. Any two principals (say P_i, P_j) that engage in a protocol run invoke algorithm $\mathrm{K}G$ (with a common input) to establish the shared key $K_{ij}(= K_{ji})$). The symbol ℓ denotes the security parameter.

Adversary A has complete control over the communication link, hence she can eavesdrop on a (polynomial) number of protocol transcripts and also delete, add, modify and replay legitimate messages exchanged by any two parties.

The adversary interacts with a polynomial number (in the security parameter ℓ) of running instances of the protocol, a.k.a. *oracles*; the r-th oracle, simulating the messages sent by P_i (to its intended partner P_j) is represented by the symbol Π_i^r, the corresponding (s-th) instance simulating P_j's responses is represented by Π_j^s. Honest party oracles behave according to the description of protocol Π. When referring to the intended partner of oracle Π_i^r we may use the notation $\Pi_i^r.pid = pid_i^r = P_j$ (and $\Pi_j^s.pid = pid_j^s = P_i$). Analogously, the variables $term_i^r, acc_i^r$ respectively indicate whether oracle Π_i^r has terminated or has accepted, meaning that an accepting oracle has output a session key while a terminating oracle has accepted but has no more messages to send nor to receive.

The session identifier sid_i^r (resp. sid_j^s) is defined as the concatenation of all the messages originating from and received by instance Π_i^r (resp. Π_j^s) in a protocol run while communicating with pid_i^r (resp. pid_j^s). Session identifiers serve to define matching conversations between instances and therefore to account for mutual authentication. More formally, we give the following definition:

Definition 1 [Partnered Instances] For any two principals P_i, P_j running protocol Π, the instances Π_i^r and Π_j^s are *partnered* if the following conditions hold:

(i) $pid_i^r = P_j \wedge pid_j^s = P_i$;

(ii) $acc_i^r = TRUE \wedge acc_j^s = TRUE$;

(iii) $sid_i^r = sid_j^s \neq NULL$

Condition (i) requires that a principal is aware of the exact identity of its partner. Condition (ii) requires that both instances have accepted, i.e. they have sent out and received all messages, according to the protocol specification, that are relevant for the computation of the session key. Observe that condition (ii) implies that two instances cannot both explicitly determine whether they are partnered or not (in particular, the instance that sends the last message will never know if her partner actually received it). Only implicit key confirmation can be achieved if the sids are included in the computation of the session key. Condition (iii) requires that the both principals have had a matching conversation.

The following correctness condition should be satisfied by any protocol.

Definition 2 [Protocol Correctness] For any two principals P_i, P_j running protocol Π in the presence of a passive adversary, the following conditions hold:

(i) $\Pi_i^r \wedge \Pi_j^s$ are *partnered*;

(ii) $sk_i^r = sk_j^s \neq NULL$

The adversary can initiate and interact with protocol instances (oracles) by asking the following queries:

- (*init*,*i*,*j*): this query sets $pid_i^r = P_j$ and activates (say, the r-th) instance Π_i^r of the protocol at principal P_i. As a result, oracle Π_i^r enters the idle state;

- (*send*,*i*,*r*,M): this query allows the adversary to send a message M to instance Π_i^r while impersonating the intended partner pid_i^r (say P_j). The response is computed exactly as specified by the protocol, with instance Π_i^r entering an expecting state. When M=*start* instance Π_i^r in the idle state is prompted to send the first message according to the protocol specification;

- (execute,i,j): this query models a passive adversary eavesdropping on a run of the protocol between honest principals P_i, P_j. The resulting transcript is given to the adversary. In principle, an execute query can be simulated by *send* and *init* queries. However, in an execute query the instances strictly adhere to the protocol specification (hence the resulting protocol executions are correct as by Definition 2.). This is opposed to send queries wherein the adversary can specify messages of her own choice (which may be indistinguishable from valid ones and hence may cause the recipient oracle to accept);

- (*reveal*,*i*,*r*): this query models exposure of the session key of the instance Π_i^r due, for example, to improper erasure after its use, hijacking of the machine running the protocol or perhaps cryptanalysis. It is applicable only to instances that have accepted;

- (*corrupt*,*i*): there is a *weak corruption model*, wherein this query returns only the long-term private key of principal Π_i^r, and a *strong corruption model* where the adversary also obtains the internal state of the instances run by P_i (and where sometimes the adversary is allowed to replace the private key of P_i). The adversary can use the compromised private key to impersonate P_i with *send* queries. We stress that the adversary does not obtain the session key as the result of a *corrupt* query on a instance Π_i^r that has accepted;

- (*test*,*i*,*r*): when the adversary A asks this query an unbiased coin b is flipped and K_b is returned. If $b=0$ then K_0 is equal to the real session key, otherwise K_1 it a random number sampled from the distribution of the session keys. The adversary must distinguish which one.

The security of a protocol is defined in the context of the following game between a challenger C and the adversary A:

(a) *Setup*: The challenger C runs algorithm $GD(1^k)$ (ℓ is the security parameter) to generate private keys for all pairs of principals P_i, P_j. The challenger also initialises all oracles (including random number generators);

(b) *Queries*: Adversary A adaptively asks queries *init, send, execute, reveal, corrupt*, but only a single *test* query to a *fresh* oracle. The challenger responds to each query as described above and also answers to random oracles queries (modeling hash functions). We stress that the adversary may diverge from the protocol specification;

(c) *Output*: Eventually, the adversary wins the game if she can distinguish whether a key obtained from the *test* query is a real session key or a random one with the same distribution (or equivalently, if the adversary correctly outputs its guess b' of the bit b chosen by the challenger when answering the *test* query).

At the end of the game the advantage of the adversary must be negligible for the protocol to be secure. In a concrete analysis the advantage is expressed as a function of the resource expenditure required to win the game.

A meaningful notion of security depends on the capabilities of the adversary, which are expressed in terms of the types of queries she is allowed to ask during the game. For example, the above game allows the modeling of *Forward Secrecy* (FS), which captures the inability of the adversary to obtain information on previously generated session keys even after principals are corrupted.

To win the game the adversary must try to guess the session key of a *FS-fresh* oracle; for an oracle (and its partner) that has not been the target of *send* queries (for the whole duration of the game above), even with the subsequent corruption of the principals, the adversary is unable to obtain any information on already established session keys. This property is formally stated as follows:

Definition 3 [FS-fresh oracle] An oracle Π_i^r is *FS-fresh* if the following conditions hold at the end of the game:

(i) $acc_i^r = TRUE$;
(ii) The adversary has not asked (*reveal,i,r*);
(iii) If a (*corrupt,i*) query was invoked then no (*send,i,r,**) was asked;
(iv) If there exists oracle Π_j^s partnered with Π_i^r then $acc_j^s = TRUE$, the adversary has not queried (*reveal,j,s*) nor (*send,j,s,**) queries were asked if a (*corrupt,j*) query was issued;

The advantage of the adversary is denoted by $Adv_\Sigma^{FS}(\ell) = |2\Pr[b = b'] - 1|$. A key agreement protocol is *FS-secure* if this advantage is maximized over all adversaries running in polynomial time *t*.

5. Security of protocol DHBT-E

In this section we prove the main result concerning the security of the protocol in the formal model of Section 4. It is straightforward to verify the correctness condition of Definition 2; if the adversary is passive and the messages exchanged by (honest) principals comply with the protocol specification then $sid_A = sid_B$ in every protocol execution, therefore the oracles are partnered and accept holding the same key (by the random oracle assumption).

The composite symmetric cryptographic key corresponds to a two factor authentication mechanism and thus affords stronger protection against attacks from an active adversary that has corrupted a party. Formally, we are only able to prove resistance against a passive adversary as far as party corruption is concerned.

We now prove the following theorem:

Theorem 1 Protocol DHBT-E is an *FS-secure* authenticated key agreement protocol assuming G is a (t,ε)-secure q-order group satisfying the CDH assumption and H is a random oracle. We have $Adv_{DHBT-E}^{\mathrm{FS}}(\ell, t, q_h, q_{re}, q_{co}, q_{ex}) \le 2^{-\|K_{ij}\|} + q_{ex}\varepsilon + 3q_s^2/q$ where *t* is the total running time of the game including the adversary's execution time, ℓ the security parameter and $q_h, q_{re}, q_{co}, q_{ex}$, respectively, the number of random oracle, *reveal, corrupt* and *execute* queries. In addition, q_n is an upper bound on the number of principals and q_s is an upper bound on the number of sessions initiated by the adversary.

Proof. For $X = g^x, Y = g^y$ the symbol DH(*X*,*Y*) represents the Diffie-Hellman secret g^{xy}. Consider an adversary A attacking the protocol in the game above (refer to Definition 3).

Let *col* denote the event that a pair of DH exponents g^a, g^b have repeated for the same two principals and *corrupt* the event that the adversary asks its *test* query to an oracle Π_i^r having $sid_i^r = P_j$, for some *i,j,r* and either (*corrupt,i*) or (*corrupt,j*) were asked.

Let *ask* be the event that, for some *i,j*, the adversary queries the random oracle H at the point (*i,j*,U,V,W,K) where W=DH(U,V), neither P_i nor P_j were corrupted nor *reveal* queries were asked of oracles Π_i^r, Π_j^s for some *r,s* during the entire course of the game and $K = K_{ij}$.

Let *Fs* denote the event that the adversary asks its *test* query to an *FS-fresh* oracle. Finally, let $succ_A$ denote the event that A outputs bit b' that is a correct guess of the bit b chosen by the challenger C when answering a *test* query in the game. With simple (but a little tedious) calculations one can see that

$$
\begin{aligned}
&|\Pr_A\left[succ_A - \frac{1}{2}\right]| \\
&=|\Pr_A[succ_A \wedge col] + \\
&\Pr_A[succ_A \wedge \overline{col} \wedge corrupt \wedge Fs] + \\
&\Pr_A[succ_A \wedge \overline{col} \wedge corrupt \wedge \overline{Fs}] + \\
&\Pr_A[succ_A \wedge \overline{col} \wedge \overline{corrupt} \wedge ask] + \\
&\Pr_A[succ_A \wedge \overline{col} \wedge \overline{corrupt} \wedge \overline{ask}] - \frac{1}{2}| \\
&\le |\Pr_A[succ_A \mid col]\cdot \Pr_A[col] + \Pr_A[\overline{col} \wedge corrupt \wedge Fs] + \\
&\frac{1}{2}(\Pr_A[col] - \Pr_A[\overline{col} \wedge \overline{corrupt}] - \Pr_A[\overline{col} \wedge corrupt \wedge Fs]) + \\
&\frac{1}{2}(\Pr_A[col] - \Pr_A[\overline{col} \wedge corrupt] - \Pr_A[\overline{col} \wedge \overline{corrupt} \wedge ask]) + \\
&\Pr_A[\overline{col} \wedge \overline{corrupt} \wedge ask] - \frac{1}{2}| \\
&\le \Pr_A[col] + \frac{1}{2}\Pr_A[\overline{col}] + \frac{1}{2}\Pr_A[corrupt \wedge Fs] + \\
&\Pr_A[col] - \frac{1}{2}\Pr_A[\overline{col}] + \frac{1}{2}\Pr_A[\overline{col}] + \frac{1}{2}\Pr_A[\overline{corrupt} \wedge ask] - \frac{1}{2} \\
&= \frac{1}{2}(3\Pr_A[col] + \Pr_A[corrupt \wedge Fs] + \Pr_A[\overline{corrupt} \wedge ask])
\end{aligned}
$$

where we used the fact that

$$\Pr_A[succ_A \mid \overline{col} \wedge \overline{corrupt} \wedge \overline{ask}] = \frac{1}{2}$$

$$\Pr_A[succ_A \mid \overline{col} \wedge corrupt \wedge \overline{Fs}] = \frac{1}{2}$$

If any two principals P_i ,P_j generate the same ephemeral random data *a,b* in two different runs of the protocol then event *col* has occurred. The (t,ε)-adversary plays the game asking a polynomial number (in the security parameter) of *execute* and *reveal* queries and stores the resulting sids and session keys until event *col* occurs. By asking the *test* query of the (fresh) oracle that has output the repeating sid the adversary can easily distinguish whether it was given a real or random key (by comparing it with the previously generated identical session key). However, the probability of event *col* occurring corresponds to an instance of the birthday problem, therefore we have $\Pr_A[col] \le q_s^2/q$.

Now consider the term $\Pr_A[\overline{corrupt} \wedge ask]$. Consider the adversary A simulating a man-in-the-middle attack in the game by issuing an (*init,i,j*) query and a (*send,i,r*,M) query with $M = g^e$ for random *e*. On receipt of the response g^a from oracle Π_i^r, if A asks the query $H(i, j, g^e, g^a, g^{ae}, K_{ij})$ then event *ask* has occurred. However, since A must guess the correct value of K_{ij} we have $\Pr_A[\overline{corrupt} \wedge ask] \le 2^{-\|K_{ij}\|}$ where $\|K_{ij}\| = \|K_{1,ij}\| + \|K_{2,ij}\|$ and the expression $\|x\|$ represents the bit-length of x.

Next we claim that $\Pr_A[corrupt \wedge Fs] \le q_{ex}\varepsilon$. To prove the later bound we construct an algorithm F that outputs DH(*X,Y*) when its input is ($X = g^x, Y = g^y$) by embedding the DH-tuples in all *execute* queries. The adversary F runs algorithm A as a subroutine and must simulate the challenger C in the game by responding to the queries of A in such a way that A's view is indistinguishable from the real game. Algorithm F runs as follows:

(1) F receives in input X,Y runs $GD(1^k)$ generating keys K_{ij} for all pairs of principals P_i, P_j;

(2) F answers the queries of A as follows:
- For (*execute,i,j*) queries (which include implicit calls to the random oracle H), F chooses random *a,b* outputs the transcript $Xg^a \| Yg^b$ and sets $sk_i^r = sk_j^s \in \{0,1\}^{\ell_1}$ (ℓ_1 is polynomially related to ℓ). Notice that, under the random oracle model, the above session keys will be indistinguishable from the real ones. Likewise, the transcript $Xg^a \| Yg^b$ is uniformly distributed just as in the real world;
- For queries of the form $H(i, j, U, V, W, K_{ij})$ the output is $v \xleftarrow{R} \{0,1\}^{\ell_1}$. If the same query was already asked before then the answer given previously is returned. If the query was asked as the result of an (*execute,i,j*) call (causing the partnered instances Π_i^r, Π_j^s to accept, for some *r, s*---see preceding item) then store (*U,V,W*) in the list L_1 (observe that the probability that L_1 is not empty is $1/q_{ex}$). Observe that by definition oracles Π_i^r, Π_j^s are both *FS-fresh;*

– Queries *init*, *reveal*, *test*, *send*, *corrupt* are answered as usual. For instances Π_i^r, Π_j^s that are the target of (*send,i,j*) queries, F eventually computes the session key by invoking the random oracle as discussed in the preceding item (using the private key K_{ij});

(3) Once the game has finished (i.e. A has terminated), F returns a random tuple (U,V,W) from L_1 (if this list is empty then return *fail*---i.e. neither *execute* nor *corrupt* queries were asked by the adversary during the game). F finds *a,b* such that $U = Xg^a$ and $V = Yg^b$ and outputs $W / X^b Y^a DH(g^a g^b)$.

If event *Fs* occurs the probability that event $succ_F$ occurs (i.e. the probability that F wins the game) is $\Pr_A[Fs]/q_{ex}$ (this is the probability that F chooses a DH-tuple in the L_1 that is from the same session that A attempts to break when event *ask* occurs). Since the running time of F is essentially the same as the running time of A the claim follows. □

Acknowledgments

The author is grateful to the anonymous reviewers for their helpful comments and suggestions.

References

[1] F. Dellutri and G. Me and M.A. Strangio, "Local Authentication with Bluetooth enabled Mobile Devices", Proceedings of IEEE International Joint Conferences ICAS'05 and ICNS'05, 2005.
[2] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution", In Proceedings of CRYPTO 1993, LNCS 773, 232-249, 1993.
[3] D. Maher, "USA Patent (No. 5,450,493): Secure Communication Method and Apparatus", Transactions of IEICE, 1993.
[4] T. Garefalakis and C. J. Mitchell, "Securing Personal Area Networks", 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, vol.3, pp.1257-1259, 2002.
[5] M. Cagalj and S. Capkun and J. P. Hubaux, "Key agreement in peer-to-peer wireless networks", In Proceedings of the IEEE (Special Issue on Cryptography and Security, 2005.
[6] S. Blake-Wilson and D. Johnson and A. Menezes, "Key Agreement Protocols and their Security Analysis", In Proceedings of the 6th IMA Int.l Conf on Cryptography and Coding, LNCS 1355, pp. 30-45, 1997.
[7] M. Bellare and D. Pointcheval and P. Rogaway, Authenticated key exchange secure against dictionary attacks", LNCS 1807, pp. 139-155, 2000.

Maurizio Adriano Strangio received his B.S. degree in computer science from the Univ. of BARI in 1989. He also received his PhD from the Univ. of "Tor Vergata", Rome in 2006. His research interest include cryptography, computer forensics and security and signal processing. He is currently a member of the ACM and IEEE.