Key agreement over a radio link *

Mario Čagalj
LCA-EPFL
CH-1015 Lausanne
Switzerland
mario.cagalj@epfl.ch

Jean-Pierre Hubaux
LCA-EPFL
CH-1015 Lausanne
Switzerland
jean-pierre.hubaux@epfl.ch

ABSTRACT

We present a simple, and yet powerful, technique for key establishment over a radio link in peer-to-peer networks. Our approach is based on the Diffie-Hellman key agreement protocol. This protocol is known to be vulnerable to the "man-in-the-middle" attack if two users involved in the protocol share no authenticated information about each other (e.g., public keys) prior to the protocol execution. In this work, we show how the natural ability of users to authenticate each other by visual and verbal contact can provide a context for secure verification of the integrity of the Diffie-Hellman parameters (e.g., Diffie-Hellman public keys). Having established such a context (e.g., by being in the vicinity of each other), even if they share no authenticated information in advance, the users can run the Diffie-Hellman protocol in a secure way: at the end of the protocol, the users will be able to check whether the Diffie-Hellman public keys they exchanged were tampered with by an attacker.

We have devoted much attention to the user-friendliness of our solution: (i) all messages in our protocol are exchanged exclusively over a radio link (neither physical contact nor an infrared link is required between the devices); (ii) the users do not have to enter any passwords. All the users have to do is to compare a short string of usual words displayed on each of their devices. We quantify the trade-off between the size of this string and the level of the provided security.

We have implemented our technique in Java. Our system is independent of the underlying operating system and can be run on a variety of personal mobile devices, including those with very limited computing power.

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. (`http://www.terminodes.org`)

EPFL-IC Technical Report No. IC/2004/16

1. INTRODUCTION

As the popularity of mobile systems such as PDAs, laptops, and mobile phones increases every day, users tend to rely more and more on them, in a growing number of situations. In this paper, we focus on the frequent case in which two persons get together (e.g., at a meeting, or in the street) and make use of their device to communicate with each other, or at least to exchange their (electronic) business cards. Clearly, the communication between these devices must be properly secured.

Very often, the two users will want the security between their devices to be peer-to-peer, thus operating independently from any authority. In practice, this means that the mobile devices must run a protocol to authenticate each other and to protect the data they exchange (to ensure confidentiality and integrity); the latter operation typically requires setting up a symmetric shared key. This key can be used to secure both immediate communications and communications taking place afterwards (e.g., when users exchange email over the Internet).

It is a common belief that peer-to-peer security is more difficult to achieve than traditional security based on a central authority; moreover, wireless communication and mobility are considered to be at odds with security. Indeed, jamming or eavesdropping is easier on a wireless link than on a wired one, notably because such mischief can be perpetrated without physical access or contact; likewise, a mobile device is more vulnerable to impersonation and to denial of service attacks.

In contrast with this widespread belief, we think that physical presence is the best way to increase mutual

trust and to exchange information in a secure way. Indeed, authentication is straightforward, as users can visually recognize each other (if they meet for the first time, they can be introduced to each other by a common friend whom they trust; or they can check each other's ID). In order to establish a shared key, they can make use of a location limited channel (e.g., physical contact or infrared [27, 8]) between their two devices. The man-in-the-middle attack is considered to be unfeasible in these conditions.

More recently, researchers have proposed solutions running exclusively on the radio link (hence they do not require a special channel such as physical contact or infrared), which increases the usability. To compensate for the much higher vulnerability of the channel, in some solutions users are required to type a password in both devices; in other solutions, they simply have to compare strings of words (the longer the string, the higher the security). The approach we describe in this paper belongs to the latter family of solutions. However, we make a very significant step further, by (i) quantifying the trade-off between security and usability, (ii) providing a detailed description of an implementation and measuring the related execution time, and (iii) explaining how this scheme can work even if the devices have very limited computing power.

The rest of the paper is organized as follows. In Section 2 we state the problem and formulate our assumptions. In Section 3 we present our solution. In Section 4 we provide a security analysis and an evaluation of the overhead. In Section 5 we describe the implementation of our proposal. In Section 6 we comment on the related work. Finally, we conclude the paper in Section 7.

2. PROBLEM STATEMENT AND SYSTEM MODEL

We consider the following problem. Two users, each equipped with a device capable to communicate over a radio link, get together wishing to establish a shared key. Although they can visually recognize each other, we assume that they share no authenticated cryptographic information (e.g., public keys or a shared secret) prior to this meeting. In addition, the users are limited to communication over a radio channel (no infrared or physical ports are available). The challenge is the following: *How do the users establish a shared key in such a scenario in a secure way?*

2.1 Challenges in radio-based systems

The Diffie-Hellman (DH for short) key agreement protocol [11] seems to be appropriate for the problem (and the set of assumptions) at hand; the DH key agreement protocol is believed to be secure against a passive adversary[1] (e.g., eavesdropping on a wireless link). Let us briefly review how the DH key agreement protocol works. To agree on a shared key, two users, Alice ($\mathcal{A}$) and Bob ($\mathcal{B}$), run a key agreement protocol as shown in Figure 1. $\mathcal{A}$ picks a random secret number x (the DH private key), and calculates the DH public key g^x, where g is a generator of a subgroup of large order. $\mathcal{B}$ does the same, that is, he calculates g^y. Finally, $\mathcal{A}$ and $\mathcal{B}$ exchange the public keys g^x and g^y and calculate the shared DH key as follows: $K = g^{xy}$.

$$\mathcal{A} \xrightarrow{g^x} \mathcal{B}$$
$$\mathcal{A} \xleftarrow{g^y} \mathcal{B}$$
$$\mathcal{A}: \quad K = (g^y)^x$$
$$\mathcal{B}: \quad K = (g^x)^y$$

Figure 1: Diffie-Hellman key agreement protocol

Unfortunately for us, the basic version of the protocol is vulnerable to an active adversary who uses a *man-in-the-middle (MITM)* attack. At first glance it may seem that mounting the MITM attack against wireless devices that communicate over a radio link and are located within the radio communication range of each other, would require a great deal of sophistication from an attacker. But this is not the case, as we will see shortly.

MITM attack by exploiting ARP vulnerabilities. The Address Resolution Protocol (ARP) [23] is a protocol used by the Internet Protocol (IP) network layer protocol to map IP network addresses to the hardware addresses used by a data link protocol. ARP-spoofing is a method where an attacker on the same radio channel (or wired) as legal users sends spoofed ARP-replies to the subject of the attack, which can fool them into sending all their packets to the attacking computer (Figure 2). In an experiment we conducted, we were able to redirect the traffic between two "legal" machines through an attacking machine, despite the fact that the two legal machines were in radio-communication range of each other. For this attack we used a collection of tools for network auditing and penetration testing, called *dsniff* [25]. We stress here that ARP-spoofing is certainly not the only way to mount the MITM attack against wireless devices.

Examples of more involved MITM attacks on Bluetooth equipped devices can be found in [14, 17].

As we saw above, the basic DH protocol run over a radio channel provides no entity authentication or key authentication (even if users' devices are in radio range

[1]This is true if and only if the Diffie-Hellman problem [20] is intractable.

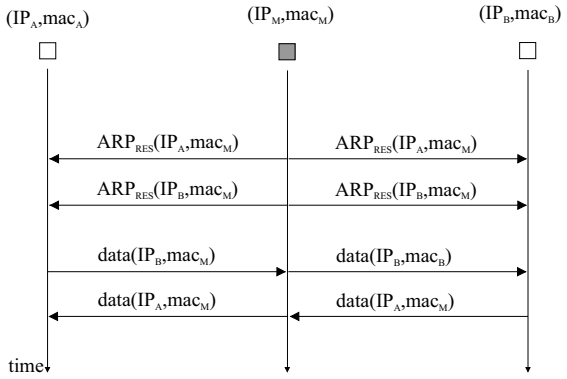

Figure 2: MITM attack by exploiting ARP vulnerability

of each other). But, if the channel used was an infrared link, the basic protocol would probably suffice. For this reason, $\mathcal{A}$ and $\mathcal{B}$ have to extend the basic protocol by an appropriate technique to make sure that their DH public keys were not tampered with by an adversary.

Integrity of Diffie-Hellman public keys. As suggested in [20, Note 12.50] and [24], an attacker may try to force the resulting key of the DH key agreement protocol into a small subset of the original key range set and then simply perform an exhaustive search over that subset. This attack works as follows. Suppose $\mathcal{A}$ and $\mathcal{B}$ choose a prime $p = Rq+1$ (R a small integer), where q is prime, and a generator g of order $p-1 = Rq$. An adversary simply intercepts the DH public keys g^x and g^y and exponentiates them with q. The secret DH key shared between $\mathcal{A}$ and $\mathcal{B}$ will be $K = g^{xyq}$, instead of $K' = g^{xy}$. Let us rewrite K as follows, $K = g^{xyq} = \alpha^{xy}$, where $\alpha = g^q = g^{(p-1)/R}$. The point is to note that α has order R (i.e., $\alpha^R = (g^{(p-1)/R})^R \equiv 1 (\mathrm{mod}\ p)$, see [20, Chapter 2.4]). Consequently, $K = \alpha^{xy}$ takes only R values. K may thus be found by exhaustive trial of R values; in practice $R = 2$.

Note that the above attack cannot be detected as an inconsistency directly from the shared DH keys of the nodes under attack (i.e., $\mathcal{A}$ and $\mathcal{B}$ still share the same key). In order to be able to detect the above attack, we should protect the integrity of the DH public keys g^x and g^y. This attack suggests that we must seek to ensure the integrity of DH public keys themselves, rather than the integrity of the established symmetric key.

Another important reason to ensure the integrity of DH public keys is based on by the following observation: people usually meet in person, and secure communication is usually needed after their first physical meeting (typically over the Internet). Clearly, in such a scenario, it is not necessary to compute the shared DH key immediately. This "expensive" computation (a modular exponentiation) can rather be postponed for some later time, when secure communication is really needed. As a consequence, if the solution to the problem of integrity checking of DH public keys is not too computationally demanding, the process of integrity checking can be carried out on computationally very "light" devices. This is very important, since, while on the move, people are often equipped with only computationally "light" devices (e.g., mobile phones, PDAs).

Yet another drawback of the integrity checking of the established DH key is that this process could potentially leak some information on the key itself; the key would not be anymore distinguishable from random.

This paper includes a possible approach to this problem of integrity checking, which exhibits properties of being light and robust at the same time.

We next introduce the system model to be used throughout the paper.

2.2 System model

We assume each user to be equipped with a small device (e.g., a PDA), which we refer to as a *node*. Each node is equipped with a radio transceiver (e.g., IEEE 802.11, Bluetooth). We also assume that each node comprises a human-friendly interface (i.e., a screen and a keyboard).

In this paper we will present our solution over the multiplicative group $\mathbb{Z}_p^*(= \{1, 2, \ldots, p-1\})$, where p is a large prime. However, all the treatment here applies to any group in which the Diffie-Hellman problem is intractable[2]. Furthermore, we assume that p and a generator g of $\mathbb{Z}_p^*$, $(2 \leq g \leq p-2)$ are selected and published. All nodes are preloaded with these values. We stress here that we could let users select and communicate to each other their own parameters p and g. However this would come at the expense of the number (and size) of messages to be exchanged between the users, while our goal is to keep the key exchange protocol as simple as possible.

Concerning the adversarial model, we assume an attacker to be *computationally bounded*: meaning that he cannot solve the Diffie-Hellman problem in $\mathbb{Z}_p^*$ or find a collision for a hash function (with security parameter[3] l_{MD}) within a polynomial time in the security parameter l_{MD}.

3. PROPOSED SOLUTION

3.1 Security Vs usability trade-off

[2]These are all groups in which it is infeasible to distinguish between quadruples of the form (g, g^x, g^y, g^{xy}) and quadruples (g, g^x, g^y, g^z) where x, y, z are random exponents.

[3]This relates to the "safe size" of the message digest output by this function.

Alice's public key g^x in hexadecimal system:

```
322B64C0F2F8FA54817D6B710B5C7C549D88E9C2E8E0
5D50A063F590982A46AD7B80A83749AC241C1F9DD391
0132341DFF9804FAF741CA7A9F420D1F14FC49062A57
B2F715FE81C20997D02FB38839095BDFD844AF07F...
179551944D632B5BD26F19B59FE2FA564C25E2454C05
8819DC0F5CFD1305EF82798532BA4ADAA6409EEBF...
```

"Fingerprint" of Alice's public key-$h(g^x)$:

```
F33B7A487FA7135DF284CE050EBA487F44E50345
```

Alice's "fingerprint" $h(g^x)$ converted to usual language words:

```
radio love war place house car mom path
  sun bit pen dog house man ball call
```

Figure 3: Visual verification of Diffie-Hellman public keys

In this section, we discuss the basic ingredients (ideas) of our approach to secure key agreement protocols when users share no authenticated information in advance.

The simplest, and yet the most reliable, way to check the validity of the exchanged DH public keys for $\mathcal{A}$ and $\mathcal{B}$, is to simply report the exchanged public keys g^x and g^y to each other and then perform a comparison of them. The comparison of the exchanged values can be performed by looking at the screen of the communicating party, or by reading aloud the values to be compared.

Although this approach provides very strong security, it is clearly impractical as can be seen from Figure 3. A possible way to make visual (and verbal) verification easier is to represent the DH public keys in a more readable form. Thus, instead of comparing the raw binary (decimal, hexadecimal etc.) information, we can replace each word of m binary digits with a meaningful item (a word or an image) from a predefined database of 2^m items (words, images); a set of m binary digits represents a unique index in the database. However, in this way the users would have to compare $\lceil \frac{l}{m} \rceil$ words (images), where l is the number of binary digits (bits) required to represent a DH public key (typically, $l = 1024$ bits). Since we should keep m small (for the sake of a node's memory), the users are still left with a large number of words (images) to be read and compared.

An alternative is to trade some robustness of visual (verbal) checking for increased usability. Thus, instead of verifying a DH public key itself (e.g., g^x), we can actually verify the hash value of it, i.e., $h(g^x)$ (like in PGP [1]). Since, in practice, hash functions produce much shorter output than DH public values, the usability of visual (verbal) checking is substantially improved. However, since DH public keys are much longer than digests output by a hash function, many different DH public keys translate to the same digest. This may give some advantage to a potential attacker. Consequently, for the hash function based approach to be useful from the security point of view, the employed hash function h should exhibit the following important properties: (i) h should be *collision resistant*; (ii) the size of a digest output by h should be no less than a specified threshold $\underline{l}$ (at the time being the recommended threshold is $\underline{l} = 160$ bits) [20].

Notice that the second property directly impacts the usability of the approach with visual (verbal) verification. Thus, for example, if we assume that our database contains 1024 different items (i.e., $m = 10$), it would require users to verify $\frac{160}{10} = 16$ items. This number is much smaller than in the case in which no hash function is used (Figure 3). The number of items to be checked can additionally be reduced by increasing the number of items in the database, i.e., increasing m. Inevitably, this results in increase in the memory requirements ($L_{avg} \cdot 2^m$, where L_{avg} is the average word size). For example, for $m = 32$ (and hence $\frac{160}{32} = 5$ items to be compared) and $L_{avg} = 4$ *Bytes*, we would need around 16 GB of memory to store the dictionary, whereas $m = 10$ requires only 4 KB.

We can also trade off memory requirements for usability. Thus, we can completely avoid the need for a database with specific items. However, in this case we are back to the representation of a given output in a number system (hexadecimal, decimal, but certainly not a binary system). As a consequence, usability deteriorates again.

We conclude that a trade-off between robustness (secu-

rity), memory requirements and usability is necessary to make the approach of visual (verbal) verification functional.

3.2 Commitment schemes

An important cryptographic building block we will be using in our protocol is message commitment. The reason we use commitments is because we want to impose the following restrictions on our users: (i) a user who commits to a certain value (a DH public key in our case) cannot change this value afterwards (the scheme is *binding*); (ii) the commitment is hidden from its receiver until the sender "opens" it (the scheme is *hiding*).

A commitment scheme transforms a value m into a commitment/opening pair (c, d), where c reveals no information about m, but (c, d) together reveal m, and it is infeasible to find $d^{'}$ such that $(c, d^{'})$ reveals $m^{'} \neq m$. Now, if $\mathcal{A}$ wants to commit a value m to $\mathcal{B}$, she first generates the commitment/openeing pair $(c_A, d_A) \leftarrow \text{Commit}(m)$, and sends c_A to $\mathcal{B}$. To open m, $\mathcal{A}$ simply sends d_A to $\mathcal{B}$, who runs $m^{'} \leftarrow \text{Open}(c_A, d_A)$. If the employed commitment scheme is "correct", at the end of the protocol we must have $m^{'} = m$.

In this paper we will make use of a collision-free hash function based commitment scheme due Halevi and Micali [13]. We call this scheme the Halevi-Micali commitment scheme [13]. This scheme is a very practical commitment scheme based solely on collision-free hashing. To commit to a message M, the sender picks at random a string x and a *universal hash function* f so that $f(x) = M$. Then the user applies the collision-free hash function h (e.g. SHA-1, which is believed to be collision free) to the random string x to get $y = h(x)$ and sends the commit string $c = (y, f)$ to the intended receiver. To open the commit string, the sender simply sends the random string x. The efficiency of this commitment scheme comes from the fact that it makes use of inexpensive hash functions only.

An example of a more computationally intensive commitment scheme is the Pedersen commitment scheme, which is perfect-hiding and computationally binding [21]. The major disadvantage of this scheme is that it requires expensive cryptographic operations like modular exponentiation. We note that Pedersen's scheme could potentially be used in scenarios where users are equipped with devices with more computational power than PDAs (e.g., laptops).

In the following section, we describe our protocol.

3.3 Protocol description

Let us first introduce the notation we will be using in the rest of the text:

DH	Abbr. "Diffie-Hellman"
$\oplus$	Bitwise "xor" operation
p	A large prime number (system parameter)
g	A generator of $\mathbb{Z}_p^*$, $(2 \leq g \leq p-2)$ (system parameter)
$(c, d) \leftarrow \text{Commit}(m)$	The commitment/opening pair (c, d) for m
$m^{'} \leftarrow \text{Open}(c, d)$	Opens the commitment with the opening key d
κ	A security parameter in our protocol ($\kappa \in \mathbb{N}$)
M	The set of $\|M\|$ distinct items (e.g., words), s.t. $\|M\|^\kappa \ll \|\mathbb{Z}_p^*\|$
iid	Abbr. "independent and identically distrib."

We first describe a basic protocol with a general commitment scheme. The employed commitment scheme should exhibit the following properties: (i) it should be perfect-hiding; (ii) it should be computationally binding [13]. We emphasize here that computationally-hiding and perfect-binding schemes would work as well. However, for the clarity of the presentation we restrict ourselves, without any loss of generality, to the first type of commitment schemes. Later in the paper (Section 4), we discuss the effects of the Halevi-Micali commitment scheme to our protocol.

Our protocol is run between two users only (e.g., Alice ($\mathcal{A}$) and Bob ($\mathcal{B}$)). The protocol for key agreement unfolds[4] as shown in Figure 4.

Both $\mathcal{A}$ and $\mathcal{B}$ select their secret exponents x and y, respectively, uniformly at random from the set $\{1, 2, \ldots, p-2\}$ and, in turn, calculate DH public keys g^x and g^y, respectively. In addition, $\mathcal{A}$ and $\mathcal{B}$ select so called *mask* elements x_m and y_m, respectively, uniformly at random from the set $\mathrm{S} \stackrel{def}{=} \{0,1\}^n$, where n is the number of binary digits in the binary representation of the largest element from the set $\{1, 2, \ldots, p-2\}$.

In the initial phase (Step 1), $\mathcal{A}$ and $\mathcal{B}$ calculate commitment/opening pairs for the tuples (x_m, g^x) and (y_m, g^y), respectively, that is:

$$(c_A, d_A) \leftarrow \text{Commit}(x_m, g^x)$$
$$(c_B, d_B) \leftarrow \text{Commit}(y_m, g^y)$$

Having calculated (c_A, d_A) and (c_B, d_B) in Step 1, $\mathcal{A}$ and $\mathcal{B}$ exchange the commitments c_A and c_B (Step 2). There is no strict order in which this exchange of messages should happen. However, notice that in most cases one party receives a commitment before the other sends its own. We will see later how this fact can help

[4]Assuming that no attack is taking place yet.

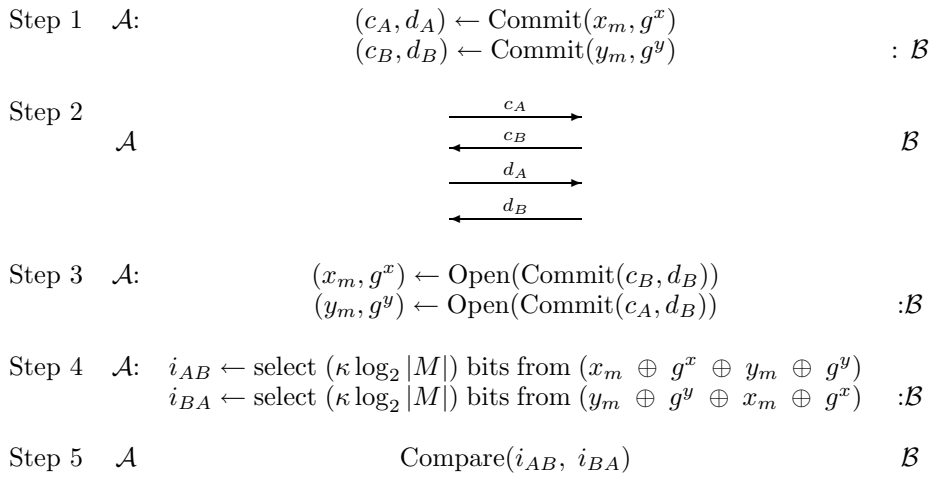

Figure 4: A commitment protocol and the integrity checking scheme integrated into the basic Diffie-Hellman key agreement protocol

us to optimize[5] our protocol.

It is only after receiving a commitment from the other side, that $\mathcal{A}$ and $\mathcal{B}$ reveal their committed values by sending the opening keys d_A and d_B, respectively, to each other. Having received the opening keys, in Step 3, $\mathcal{A}$ and $\mathcal{B}$ open the committed values (y_m, g^y) and (x_m, g^x) by running $\text{Open}(\text{Commit}(c_B, d_B))$ and $\text{Open}(\text{Commit}(c_A, d_A))$, respectively. At this stage $\mathcal{A}$ and $\mathcal{B}$ should make sure that their commitments (i.e., DH public keys) have not been modified by an attacker. As we discussed earlier, the most reliable way to perform this is to compare each and every digit (decimal, hexadecimal, etc.) of their respective DH public keys[6]. But, as we argued earlier, this is a rather cumbersome and potentially error-prone task. Our approach to this verification problem is as follows.

In Step 4, $\mathcal{A}$ and $\mathcal{B}$ first "xor" all the DH public keys and their mask elements, i.e., $(x_m \oplus g^x \oplus y_m \oplus g^y)$ and $(y_m \oplus g^y \oplus x_m \oplus g^x)$, respectively. Then $\mathcal{A}$ and $\mathcal{B}$ simply select $\kappa \log_2 |M|$ bits from the result of the above "xor" operations to variables i_{AB} and i_{BA}, respectively. Since the result of these "xor" operations is a random number from the set S (we prove this later), the selection criterion does not really matter (e.g., taking the first $\kappa \log_2 |M|$ bits is a possible strategy). Note here that we assume $|M| = 2^l$, $l \in \mathbb{N}$, with only negligible loss of generality. Also note that $\kappa \log_2 |M|$ should be less than the number of binary digits in the representation of the prime p. Typically, $\kappa = 4$ and $\log_2 |M| = 10$, while $\log_2 |Z_p^*| \approx 1024$.

[5]Meaning, to reduce the number of messages to be exchanged between the two parties.

[6]If this were the case, then we would need no commitment scheme in our protocol.

Finally, in Step 5, $\mathcal{A}$ and $\mathcal{B}$ take the binary representation of i_{AB} and i_{BA}, respectively, and then simply break it into κ l-bits binary words, where $l = \log_2 |M|$. These κ binary words are in turn used as indices into the set M. If the κ ordered items, retrieved from the common set M, match, users $\mathcal{A}$ and $\mathcal{B}$ accept the exchanged DH public keys as being authentic and simply calculate the shared DH key $K = g^{xy} \mod p$. Note here that κ specifies the number of items to be compared by users $\mathcal{A}$ and $\mathcal{B}$ and thus directly influences the usability of our approach.

In the following section, we assess the security of our protocol. We also study the overhead induced to the basic DH protocol.

4. ASSESSMENT

4.1 Security analysis

In this study, we assume that all messages exchanged over a radio link pass through an attacker (which we call Mallory ($\mathcal{M}$), Figure 5). Thus $\mathcal{M}$ can do whatever he wants (and is capable of doing) with messages. For example, $\mathcal{M}$ can drop an arbitrary message, he can modify messages, insert a new message etc. However, we assume that the attacker is computationally bounded (as described in Section 2.2). We believe that this model represents a very strong attacker for the problem at hand.

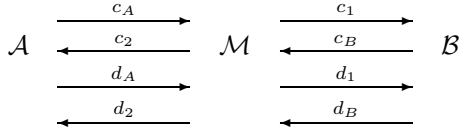

Figure 5: Attacker model

We next study possible attacking strategies for $\mathcal{M}$. Let us first introduce the following notation. Let (i, j) be a tuple of two binary digits with the following meaning:

$$i = \begin{cases} 1, & \mathcal{M} \text{ tampers with } \mathcal{A}\text{'s messages} \\ 0, & \mathcal{M} \text{ passes through } \mathcal{A}\text{'s messages.} \end{cases}$$

$$j = \begin{cases} 1, & \mathcal{M} \text{ tampers with } \mathcal{B}\text{'s messages} \\ 0, & \mathcal{M} \text{ passes through } \mathcal{B}\text{'s messages.} \end{cases}$$

With the above notation we can define the following possible[7] attacks by $\mathcal{M}$:

(0,1)-attack In this attack, $\mathcal{M}$ simply wants to deceive $\mathcal{A}$ into believing that $\mathcal{M}$ is actually $\mathcal{B}$. We term this attack a "half man-in-the-middle" ($\frac{1}{2}$-MITM) attack.

(1,0)-attack This attack is exactly the same as (0,1)-attack, but with swapped roles between $\mathcal{A}$ and $\mathcal{B}$.

(1,1)-attack In this attack, $\mathcal{M}$ aims at fooling both, $\mathcal{A}$ and $\mathcal{B}$, into believing that $\mathcal{M}$ is actually $\mathcal{B}$ and $\mathcal{A}$, respectively. We call this attack simply the "man in the middle" (MITM) attack.

Eavesdropping attack. Note that the (0,0)-attack actually corresponds to an eavesdropping attack. Since we assume that $\mathcal{M}$ cannot solve the Diffie-Hellman problem, we do not take the (0,0)-attack as being part of $\mathcal{M}$'s attacking strategy.

Mirroring attack. In this attack, $\mathcal{M}$ simply "mirrors" all messages he receives back to the messages' originators. Following the notation in Figure 5, $\mathcal{M}$ sets $c_2 = c_A$, $d_2 = d_A$, $c_1 = c_B$ and $d_1 = d_B$. Clearly, the verification phase will be successful, since $(x_m \oplus g^x \oplus x_m \oplus g^x) = (y_m \oplus g^y \oplus y_m \oplus g^y) = 0$, even though the keys $K = g^{xx}$ and $K' = g^{yy}$ are not the same. Fortunately, this attack is easy to detect by simply to checking if the result of the above "xor" operation is equal 0. The probability that this happens is extremely small (as we will see shortly, this probability can be as small as 2^{-1024}). Moreover, this attack gives no advantage to $\mathcal{M}$ (i.e, $\mathcal{M}$ still does not learn g^{x^2} or g^{y^2}). In the rest of the paper, we will assume that the detection of the mirroring attack is always performed.

We next justify the need for mask elements x_m and y_m. For this purpose, first note that value $c = a \oplus b$ is unique for fixed a and b. In the context of our protocol, this means that $x_m \oplus g^x$, for example, uniquely maps $\mathcal{A}$'s DH public key g^x to the set $\mathrm{S} = \{0,1\}^n$, for fixed x_m and g^x. As a consequence, checking the integrity of $g^x \in \mathbb{Z}_p^* \backslash \{1\}$ is equivalent to checking the integrity of $(x_m \oplus g^x) \in \mathrm{S}$ for the fixed x_m. Note that g^x and x_m do not necessarily belong to the same set, i.e., $x_m \in \mathrm{S}$. Hence, for x_m chosen uniformly at random from S, we have that the probability of selecting any value from this set is equal to 2^{-n}, that is, the probability of an arbitrary bit in the binary representation of the selected number being 0 (or 1) is equal to 2^{-1} (S comprises all the n-bits vectors). Note that this does not apply to the set $\mathbb{Z}_p^* \backslash \{1\}$, i.e., the probability of an arbitrary bit, of the randomly selected number from $\mathbb{Z}_p^* \backslash \{1\}$, being 0 (or 1) $\neq 2^{-1}$ (e.g., for $\mathbb{Z}_5^* \backslash \{1\} = \{010_{(2)}, 011_{(2)}, 100_{(2)}\}$, the probability of the first bit of a randomly selected 3-bit vector being 0 (1) is 2/3 (1/3)). We can now formulate the following straightforward lemma.

LEMMA 1. *Let X_1, $X_2 \in \{0,1\}^n$ be two independent random variables such that the distribution of X_1 is uniform (i.e., $Pr\{X_1 = x\} = 2^{-n}, \forall x \in \{0,1\}^n$) and the distribution of X_2 is unknown. Then, the distribution of $X_1 \oplus X_2$ is uniform.*

Proof:

$$\begin{aligned} Pr\{X_1 \oplus X_2 = x\} = & \\ &= \textstyle\sum_{\forall y} Pr\{X_1 = x \oplus y\} Pr\{X_2 = y\} \\ &= 2^{-n} \textstyle\sum_{\forall y} Pr\{X_2 = y\} \\ &= 2^{-n} \end{aligned}$$

□

Consequently, the probability of an arbitrary bit of $x_m \oplus g^x$ being 0 (or 1) is 2^{-1}. This is exactly why in Step 4 of our protocol (Figure 4) we are indifferent about the selection strategy of $\kappa \log_2 |M|$ bits to be compared by users $\mathcal{A}$ and $\mathcal{B}$.

We are now ready to prove the following theorem.

THEOREM 1. *Assuming that the employed commitment scheme is perfect-hiding and computationally binding, the probability that a computationally bounded attacker successfully performs any (i, j)-attack, with $i, j \in \{0,1\}$ and $i + j \neq 0$, is no more than $\frac{1}{|M|^\kappa}$.*

Proof: Let (z_{m1}, g^{z_1}) and (z_{m2}, g^{z_2}) be $\mathcal{M}$'s fake DH public keys and mask elements in the case of (1,0)-attack and (0,1)-attack, respectively. We also define two variables v, v' as follows:

$$\begin{aligned} v &= (x_m \oplus g^x \oplus z_{m2} \oplus g^{z_2}) \\ v' &= (y_m \oplus g^y \oplus z_{m1} \oplus g^{z_1}) \end{aligned}$$

In the first part of the proof, we will show that variables v and v' are two *iid* uniform random variables over the

[7]Without loss of generality, we will consider that $\mathcal{A}$ sends her commitment before $\mathcal{B}$; in practice, however, this order does not have any impact on the robustness of the protocol (the protocol is symmetric).

set S $= \{0,1\}^n$ for any attacking strategy chosen by $\mathcal{M}$. Then, in the second part, we will use this fact to evaluate the probability of a successful attack.

1st part. In this part we show that v and $v^{'}$ are indeed two *iid* uniform random variables over the set S for any possible attacking strategy chosen by $\mathcal{M}$.

(0,1)-attack Recall that in this attack $\mathcal{M}$ tampers with $\mathcal{B}$'s DH parameter and mask element only. Thus we have $v = (x_m \oplus g^x \oplus z_{m2} \oplus g^{z_2})$, while $v^{'} = (y_m \oplus g^y \oplus x_m \oplus g^x)$ (i.e., $g^{z_1} = g^x$ and $z_{m1} = x_m$, although Mallory learns neither x nor x_m). It is easy to see that $v^{'}$ in this case is a random variable uniformly distributed over the set S. This follows from the fact that x_m and y_m are two random variables and uniformly distributed over S. Moreover, $\mathcal{A}$ and $\mathcal{B}$ chose x_m and y_m independently of each other.

Let us now check variable v. Notice first that neither $\mathcal{A}$ nor $\mathcal{B}$ reveal their contributions (g^x, x_m) and (g^y, y_m), respectively, before sending her/his own commitment to and receiving a commitment from the other side (which can be $\mathcal{M}$). Since the employed commitment scheme is perfect hiding, the receiver of a commitment gains no information about a value committed to by the sender before the sender actually opens the commitment. Thus, if $\mathcal{M}$ wants to learn about any tuple (DH public key, mask element) in order to adjust his DH public key (or mask element) accordingly, he has two choices: (i) $\mathcal{M}$ waits until the party in question gets a commitment from the other legal party; (ii) $\mathcal{M}$ sends out his own faked commitment to the party whose DH public key (or mask element) $\mathcal{M}$ wants to learn. Since the employed commitment scheme is computationally binding (and $\mathcal{M}$ is computationally bounded), in neither of these two cases will $\mathcal{M}$ be able to change the commitment received by the party in question.

The best that $\mathcal{M}$ can do is to try to send his own fake commitment c_2 to $\mathcal{A}$ while intercepting and dropping the one of $\mathcal{B}$. However, this means that he has to choose his DH public key (and the mask element) independently of $\mathcal{A}$; the commitment scheme is binding and thus $\mathcal{M}$ cannot change the value he committed to once $\mathcal{A}$ reveals x_m (g^x). Now, since $v = (x_m \oplus g^x \oplus z_{m2} \oplus g^{z_2})$ and x_m is a uniform random variable over S, v must be a uniform random variable over S as well. This follows from Lemma 1 and the fact that x_m is chosen randomly and independently of g^x, z_{m2} and g^{z_2}). Finally, since x_m and y_m are chosen independently of each other, then it must be that v and $v^{'}$ are two independent random variables. Q.e.d.

(1,0)-attack The fact that v and $v^{'}$ are two *iid* uniform random variables in this attack follows trivially from the (0,1)-attack.

(1,1)-attack In this attack, $v = (x_m \oplus g^x \oplus z_{m2} \oplus g^{z_2})$ while $v^{'} = (y_m \oplus g^y \oplus z_{m1} \oplus g^{z_1})$. Note that in order to perform this attack, $\mathcal{M}$ has to successfully mount the combination of (0,1) and (1,0)-attacks. We showed above that in both (0,1) and (1,0) attacks, v and $v^{'}$ are two *iid* uniform random variables. Consequently, v and $v^{'}$ are *iid* uniform random variables in the (1,1)-attack as well.

Thus, any attempt by $\mathcal{M}$ to tamper with DH public keys will result in two *iid* uniform random variables over the set S $\{0,1\}^n$, namely, v and $v^{'}$. Consequently, the best $\mathcal{M}$ can hope to achieve is to provoke a *collision* between $\kappa \log_2 |M|$ bits of v and $v^{'}$, selected in Step 4 of the protocol. It is exclusively in this case that κ indices generated from v and $v^{'}$ (as specified in Step 4 of the protocol) will match.

Now we are ready to evaluate the probability of successfully performing a (i,j)-attack, with $i,j \in \{0,1\}$ and $i + j \neq 0$, that is, the probability that $\kappa \log_2 |M|$ chosen bits of v and $v^{'}$ match.

2nd part. Let us introduce the following three events:

$$\begin{aligned} A &= \{ \text{ MITM attack attempted } \} \\ A_s &= \{ \text{ MITM attack successful } \} \\ C &= \{ \kappa \log_2 |M| \text{ bits of } v = \kappa \log_2 |M| \text{ bits of } v^{'} \}. \end{aligned}$$

In addition, let b_i^v denote the ith bit to be compared in Step 5 of Figure 4, extracted from the random variable v. As we argued above, the only hope for $\mathcal{M}$ is to wait for the event C to happen; whatever $\mathcal{M}$ does, v and $v^{'}$ are two *iid* random variables over S $\{0,1\}^n$. In other words, we have the following satisfied:

$$\begin{aligned} Pr\{A_s|A\} &= Pr\{C\} \\ &= \prod_{i=1}^{\kappa \log_2 |M|} Pr\{b_i^v = b_i^{v^{'}}\} \\ &= \prod_{i=1}^{\kappa \log_2 |M|} \frac{1}{2} \\ &= \frac{1}{2^{\kappa \log_2 |M|}} \\ &= \frac{1}{|M|^\kappa} \end{aligned}$$

□

An interesting aspect of the proposed protocol is that an attacker is given only one chance to guess the DH public key (or/and mask value) of $\mathcal{A}$ and $\mathcal{B}$. For this reason, the birthday attack against $\kappa \log_2 |M|$ bits to be compared is not a concern here.

Example: For $\kappa = 4$ (items to be verified) and $|M| = 1024$ (stored objects in the dictionary), the probability of $\mathcal{M}$ successfully mounting an attack is at most $\frac{1}{|M|^\kappa} =$

$\frac{1}{2^{40}} < 10^{-12}$. As pointed out above, an attacker has only a single chance to guess the string of 40 random bits.

4.2 Optimization and overhead estimation

As we already stated, by exploiting the intrinsic asymmetry of the unfolding of our protocol, we can reduce the number of messages exchanged between the two users. In most cases, one of the users receives a commitment before the other one sends his own[8].

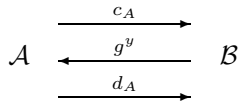

Figure 6: Optimization of the basic protocol (only messages exchanged over a radio link are shown)

Without any loss of generality, we assume that $\mathcal{A}$ sends her commitment first. Since the commitment scheme is computationally binding, having received the commitment from $\mathcal{A}$ (or $\mathcal{M}$ in case of an attack), $\mathcal{B}$ does not have to hide his DH public key. Thus, $\mathcal{B}$ simply replies with his DH public key g^y. Note that the bits to be compared as in Step 5 (Figure 4) of the symmetric version, are now obtained from the following value: $v = (x_m \oplus g^x \oplus g^y)$. Since x_m is selected uniformly at random from $S = \{0,1\}^n$, and independently of g^y (or g^{z_2}, potentially selected by Mallory), it follows from Lemma 1 that v is uniformly distributed over the set $S = \{0,1\}^n$. Consequently, Theorem 1 applies to this optimized version of the key agreement protocol as well.

As can be seen from Figure 6, this optimization reduces the number of necessary messages to only three. This, in conjunction with the computationally inexpensive Halevi-Micali commitment scheme [13], makes our approach very efficient and practical.

To assess the overhead due to our extension to the basic DH protocol, we begin with the Halevi-Micali commitment scheme. Let k be the security parameter[9] of the employed commitment scheme and r the length of the message (in bits) being committed to. According to [13], the length of commitment c in the Halevi-Micali scheme is equal to $O(k)$ (typical k equals 128 bits), i.e., the length of c is independent of r (e.g., $r = 2 \times 1024$)). In terms of local computation, the Halevi-Micali scheme takes: 1 collision-free hashing of r-bits message; 1 collision-free hashing of $O(k)$-bit string; 1 universal-hashing of $O(k)$-bit string. Finally, in Step 4 of Figure 4, each device should still perform three computationally cheap "xor" operation.

As far as the communication cost is concerned, in the optimized version of our protocol (Figure 6), $\mathcal{A}$ transmis $O(k)$-bits in her first message (the commit stage) and $r + O(k)$ bits in her second message (the de-commit stage), whereas $\mathcal{B}$ transmits only his DH public key g^y (i.e., typically 1024 bits). Note that in terms of the number of packets to be exchanged between $\mathcal{A}$ and $\mathcal{B}$, this amounts to only 3 packets (even at the MAC layer).

Based on the above analysis, we conclude that our protocol induces modest communication and computation overhead to the basic DH key agreement protocol.

5. IMPLEMENTATION

We have implemented a prototype of the described approach to the key agreement in Java [4]. The key agreement scheme we actually implemented is shown in Figure 7. We use the SHA-1 hash function to construct the commitment scheme; to commit to a value m, the sender simply runs SHA-1(m) and sends the obtained value to the receiver. In the de-commit phase, the sender simply sends m, while the commitment receiver checks if $m \stackrel{?}{=}$ SHA-1(m). We believe that such a construction of the commitment scheme is still sufficiently secure, although we cannot provide mathematical evidence to backup this statement. However, if a provable security is required, one can always resort to the Halevi-Micali commitment scheme at the expense of one additional collision-free hashing and one universal-hashing; evaluating a universal-hash function is typically cheaper than evaluating a collision-free hash function [13]. At the end, an adversary is quite restricted in terms of the available time to analyze the messages exchanged between two legal parties; it is quite obvious that the legal parties will not wait forever to receive messages from each other (this delay will be in the order of milliseconds).

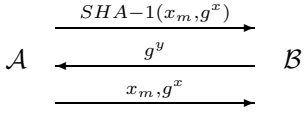

Figure 7: Implemented key agreement protocol

In our design, we use a simple and well established distributed programming *client/server* paradigm. At application startup, users are offered two options: either (i) to run the client process or (ii) to run the server process. Note in this implementation, that two users must agree on the roles prior to the protocol execution; as future work we plan to automate this procedure. In the current implementation, the client and the server learn about each other via their IP addresses. In terms

[8]The users cannot receive and transmits simultaneously; at least not with CSMA/CA protocols (e.g., IEEE 802.11).

[9]The security parameter may control the success probability of the commitment sender in changing her message after having committed to it, as well the probabilistic advantage the commitment receiver may get about the message from its commitment [13].

of users' involvement, this amounts to the server user simply waiting after having started the server process. The client user, on the contrary, is more involved; he has to enter the IP address of the server's machine into a particular text field, and in turn initiate the protocol by clicking on a given button. From this point on, the application takes over the control and finishes the protocol. The two users then compare their displays. The application state diagram is shown in Figure 8.

As can be seen in Figure 8, all cryptographic operations (DH key generation, key agreement) are done on the spot. Clearly, such an implementation design is suitable for computationally powerful devices (i.e., laptops and state of the art handheld computers). However, for computationally "light" devices, the design strategy should be appropriately adapted. We will study this case later in the section.

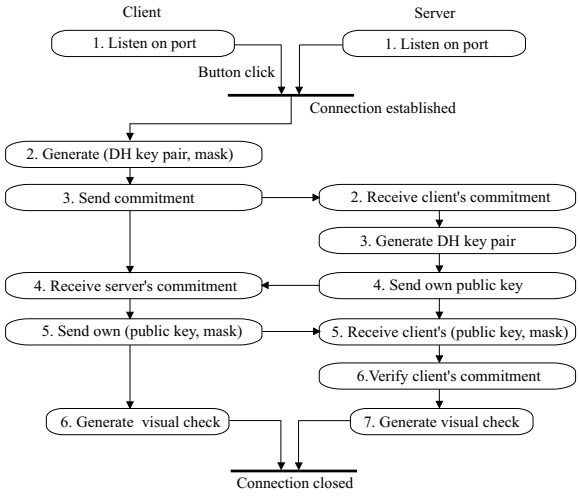

Figure 8: The application state diagram

At the network layer, our application uses the reliable transport protocol TCP. This is to avoid dealing with packet losses, acknowledgments and similar issues at the application layer. Such a choice is an important aspect in our implementation, since all the functionality for the proposed key agreement protocol remains in the application layer.

Prior to using the DH key agreement protocol, users have to agree on the same numerical values for the modulus (p) and the key basis (g). In our implementation, these parameters are preconfigured; we use the Simple Key-management for Internet Protocols (SKIP) specifications [7]. For all cryptographic operations, including generation of DH key pairs, *Java Cryptography Extension (JCE) 1.2.2* [4] is used. This cryptographic library also provides a framework and implementations for encryption and Message Authentication Code algorithms. JCE 1.2.2 has a provider-based architecture. Providers signed by a trusted entity can be plugged into the JCE 1.2.2 framework, and new algorithms can be added seamlessly. For our purpose, we utilize the *SunJCE* provider included with the JCE 1.2.2 release.

After the last message has been delivered, the application composes and displays the set of 5 words to be compared (visually or verbally) by the users. Each word encodes 8 bits, while the whole set encodes 40 bits. For this purpose, we borrowed a part of the dictionary of short words (up to four characters) from RFC 2289 [2]. Our pre-configured dictionary comprises 256 different words as opposed to 2048 words in the case of RFC 2289.

Finally, on each side the application computes the shared DH key. In the current implementation, we are not concerned with the key management issues, however, we will include this important aspect in a future version.

Performance

We evaluated the performance of cryptographic primitives of the JCE 1.2.2 cryptographic library, which is used in our implementation. Particularly, we assessed the performance (measured in milliseconds) of the DH key pair generator and the DH shared key generator, since these are the most time consuming operations in our protocol. To estimate the time needed to generate a single DH key pair, we use the following Java code:

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
random.setSeed(seed);
kpg.initialize(PARAMETER_SPEC, random);
KeyPair kp = kpg.generateKeyPair();
```

Note from the above code that the measured time also accounts for side operations needed for the DH key pair calculation. Similarly, the code to estimate the time needed to calculate the shared DH key is given below.

```
KeyPair kp = (KeyPair) session.get(CLIENT_KEY_PAIR);
KeyAgreement = ka.init(kp.getPrivate());
kp = (KeyPair) session.get(SERVER_KEY_PAIR);
Key key = ka.doPhase(kp.getPublic(),true);
byte[] secret = ka.generateSecret();
```

Note here that the measured time includes the time it takes to retrieve private and public DH keys from the storage structure called `session` (a Java `Hashtable` in our implementation).

In all experiments we use 1024-bit prime modulus p. The experiments are performed on a 700MHz Pentium III Windows 2000 PC. The average and the standard deviation of the elapsed time are calculated for 25 runs and are presented in Table 1.

Table 1: Performance of the Diffie-Hellman key pair and the Diffie-Hellman shared key computations (expressed in milliseconds)

Operation	avg	stdv
DH key pair calculation	149.29	8.77
DH shared key calculation	118.41	5.67

An important message from the numerical values in Table 1 is that the delay due to the demanding cryptographic operations does not deteriorate the usability of our approach. Even if all cryptographically demanding operations are done on the spot, this is not a problem from the usability point of view. We suspect that an optimized implementation is C/C++ would yield even better results.

To tailor our protocol for computationally "lighter" devices than our test-bed machine, we should first understand their limitations. In [28], Wong et al. evaluated several cryptographic system libraries for Palm devices. The platforms they have tested these libraries was a 2MB Palm V and 8MB PalmIIIc running on a 16MHz and a 20MHz microprocessor. In terms of processing power, these devices are indeed "light" compared to our test-bed machine. Notice, however, that these Palm devices are still memory rich. For us, the most important results are the ones related to the measured performance of hash functions and modular exponentiations. Thus, for example, the throughput achieved with SHA-1 for a message size of 2KB is around 17,429 Bps. On the other hand, a 512-bit modular exponentiation takes around 96.91 seconds.

The limitations of computationally "light" devices suggest the following protocol design strategy. Prior to engaging in any communication, we simply pre-load the devices with an adequate number of pre-computed DH key pairs and corresponding mask values (of course, these should still be generated at random (e.g., on a PC)). Recall that we are not limited in memory resources. When two users engage in key establishment, they follow the steps as shown in Figure 8, with the only difference that: (i) instead of generating new DH key pairs, the users take the preloaded ones and (ii) the users do not generate the shared DH key at the protocol termination; they postpone this for some later time when secure communication is needed. In this adapted version of our protocol, the users only have to perform computationally cheap hash function evaluations and "xor" operations. Note that the protocol design strategy just described is adapted to scenarios where secure communication is not needed on the spot. As we already observed in Section 2, this will often be the case; people usually meet in person, and secure communication is usually needed after their first physical meeting.

Thanks to the high adaptivity and simplicity of our protocol, we are able to meet diverse requirements from different operational scenarios.

6. RELATED WORK

The problem of key establishment is a very active area of research.

Stajano and Anderson propose the *resurrecting duckling* security policy model [27, 26], in which key establishment is based on the physical contact between communicating parties (their PDAs). A physical contact acts as a *location limited channel*, which can be used to transmit a key (or a secret) in plaintext. Thus, no cryptography is required at this stage[10]. The potential drawback of this approach is realization of the physical port. Moreover, realization of a physical contact could be a bit cumbersome with bulky devices (e.g., laptops).

An approach inspired by the resurrecting duckling security policy model is proposed by Balfanz et al. [8]. In this work, the authors go one step further and relax the requirement that the location limited channel has to be secure against passive eavesdropping; they introduce the notion of a *location-limited channel* (e.g., an infrared link). A location-limited channel is used to exchange pre-authentication data and should be resistant to active attacks (e.g., man-in-the-middle). Once pre-authentication data are exchanged over a location-limited channel, users switch to a common radio channel and run any standard key exchange protocol over it. Possible candidates for a location-limited channel include: physical contact, infrared, and sound (ultrasound) [8]. The disadvantage of this approach is that it may be a bit cumbersome (i.e., requires a high degree of precision by user). In addition, the infrared link itself is not well studied in the context of secure communication. Actually, the technique we propose in this paper could be a supplement to the infrared link.

Asokan and Ginzboorg propose another solution based on a shared password [6]. They consider the problem of setting up a session key between a group of people (i.e., their computers) who get together in a meeting room and who share no prior context. It is assumed that they do not have access to public key infrastructure or third party key management services. The proposed solution is the following. A fresh password is chosen and shared among those present in the room (e.g., by writing it on a sheet of paper or a blackboard). The shared password is then used to derive a strong shared session key. This approach requires users to type the chosen password in their personal devices. The major disadvantage

[10]This means that the location limited channel should be resistant to eavesdropping, a reasonable assumption in this case.

of this approach is that users should prevent the password leakage (which may be tricky for the problem at hand). In addition, weak passwords are vulnerable to dictionary attacks.

It is well known that IT security systems are only as secure as the weakest link in these systems. In most IT systems the weakest link are the users themselves. People are slow and unreliable when dealing with meaningless strings, and, they have difficulties remembering strong passwords. In [22], Perrig and Song suggest using hash visualization to improve the security of such systems. Hash visualization is a technique that replaces meaningless strings with structured images. However, images (and patterns) used in this system are too complex, which in the end may lead to error-prone verification. Using such complex image patterns is not well suited for computationally "light" devices. Moreover, this technique works only if the users are at the same location.

It was recently brought to our attention that Dohrmann and Ellison [12] proposed a method for key verification that is similar to our approach; this method is based on converting key hashes to readable words or to an appropriate graphical representation. However, it seems that users are required to compare too many words (or graphical objects); this task could take them as much as 24 seconds according to [12]. This time is significantly reduced when the graphical representation is used. However, as we already pointed out, comparing images (as suggested in [12]) remotely (e.g., a phone) seems to be a very painful (and time consuming) task. In addition, Dohrmann and Ellison provide no security analysis of their approach. In contrast, in our paper, we show that even if the list of items to be compared is very short (e.g., 5 words of up to 4 characters, as in our implementation), we can still provide a satisfactory level of security (e.g., the probability that the MITM attack goes undetected is as small as 10^{-12}).

In US patent no. 5,450,493 [19], Maher presents several methods to verify public keys exchanged between users. The first method described in [19] is the most relevant one for the problem we consider in this paper; other methods are based on certificates and/or shared secrets[11]. Thus, $\mathcal{A}$ and $\mathcal{B}$ first perform the DH key exchange protocol and in turn report to each other values $a = f(K_A)$ and $b = f(K_B)$, where K_A and K_B are the shared DH keys as computed by $\mathcal{A}$ and $\mathcal{B}$, respectively, and f is a compression function (i.e., f maps a key to 4-digit hex vectors [19]). Unfortunately, this technique has a flaw, which was discovered by Jacobsson [15]. The problem with Maher's technique is the following. An attacker $\mathcal{M}$, who knows f, first generates his secret exponents x_1 and x_2 and the corresponding public keys g^{x_1} and g^{x_2}. Since $\mathcal{M}$ knows that $\mathcal{A}$ and $\mathcal{B}$ will compare $f(g^{x_A x_2})$ and $f(g^{x_B x_1})$, he checks if $f(g^{x_A x_2}) = f(g^{x_B x_1})$. If this is the case, $\mathcal{M}$ sends g^{x_2} to $\mathcal{A}$, and g^{x_1} to $\mathcal{B}$. If these are not equal, $\mathcal{M}$ generates new values for x_1 and x_2 and repeats the above procedure. Since f outputs a very short string (4-digit hex vector [19]), $\mathcal{M}$ will find a collision after a relatively low number of attempts (thanks to the birthday paradox).

[11]Recall that we assume that users share no authenticated information about each other, prior to engaging in a key exchange.

Motivated by the flaw in [19], Jakobsson [15] and Larsson [18] proposed two solutions. However, both solutions are based on a temporary secret shared between the two users (thus, for example, SHAKE stands for *Shared key Authenticated Key Exchange*). In our paper, we consider the same problem but in a more demanding setting, as we assume that the users share no secret key prior to the key exchange.

We have to mention other key-exchange protocols, proposed primarily for the use in the Internet: IKE [3], JFK [5], SIGMA [16]. All these protocols involve authentication by means of digital signatures, which clearly does not fit the problem we study here. We also have to mention the work of Corner and Noble [9, 10], who consider the problem of transient authentication between a user and his device.

7. CONCLUSION

In this paper, we have provided a solution to the fundamental problem of key agreement over a radio link. As user-friendliness is extremely important for the acceptance of any security scheme, we have minimized the burden on the user: there is no need of physical contact or of infrared communication between the devices; moreover, the contribution of the user is limited to the comparison of a short string of usual words displayed on each of their devices. We have shown that the proposed scheme can work even if the devices have very limited computing power.

As we have mentioned, the research community recognized the relevance of this problem several years ago and has proposed a number of solutions. However, to the best of our knowledge, the solution we propose is the most flexible one; moreover, it is the only one for which the trade-off between security and usability is quantified.

The scheme proposed in this work is not restricted to wireless networks. For example, the exchange of data could take place over the Internet and the users could compare the strings by speaking on the phone (assuming that they can authenticate each other in this way).

From the security point of view, this new setting for the key exchange reveals an interesting intrinsic advantage: we do not have to pay a special attention to users' iden-

tities anymore, as is the case when certificates are used. Upon the key exchange protocol termination, users are free to bind an arbitrary string (or name) with the key just verified.

In terms of future work, we intend to study the simultaneous key agreement between more than two users. We also plan to provide an implementation for PDAs.

8. ACKNOWLEDGMENTS

We thank Levente Buttyán, Serge Vaudenay, Markus Jakobsson, Adrian Perrig and Gildas Avoine for helpful discussions and comments.

9. REFERENCES

[1] PGP. http://www.pgpi.org/.

[2] RFC 2289 - A One-Time Password System. http://www.ietf.org/rfc/rfc2289.txt?number=2289.

[3] RFC 2409 - The Internet Key Exchange (IKE). http://www.ietf.org/rfc/rfc2409.txt?number=2409.

[4] Sun Microsystems Inc. http://sun.java.com.

[5] W. Aiello, S. M. Bellovin, M. Blaze, R. Canettia, J. Ioannidis, A. D. Keromytis, and O. Reingold. Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols. In *Proceedings of ACM Computer and Communications Security (CCS) Conference*, pages 48–58, Washington, DC, 2000.

[6] N. Asokan and P. Ginzboorg. Key Agreement in Ad-hoc Networks. *Computer Communications*, 23(17):1627–1637, November 2000.

[7] A. Aziz, M. Patterson, and G. Baehr. Simple Key-Management for Internet Protocol (SKIP). In *INET'95*. http://www.isoc.org/HMP/PAPER/244/abst.html.

[8] D. Balfanz, D. Smetters, P. Stewart, and H. Wong. Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In *Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS)*, 2002.

[9] M. Corner and B. Noble. Zero-interaction authentication. In *Proceedings of MobiCom'02*, Atlanta, Georgia, September 2002.

[10] M. Corner and B. Noble. Protecting applications with transient authentication. In *First ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, San Francisco, CA, May 2003.

[11] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 1976.

[12] S. Dohrmann and C. Ellison. Public-key Support for Collaborative Groups. In *Proceedings of the 1st Annual PKI Research Workshop*, 2002.

[13] S. Halevi and S. Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In N. Koblitz, editor, *Advances in Cryptology–CRYPTO '96*, pages 201–215. Lecture Notes in Computer Science, Springer-Verlag, 1996.

[14] M. Jacobsson and S. Wetzel. Security weaknesses in Bluetooth. Technical report, Bell Labs, 2001.

[15] M. Jakobsson. Payments and Diffie-Hellman key exchange (presentation slides). http://www.rsasecurity.com/rsalabs/-staff/bios/mjakobsson/teaching/index.html.

[16] H. Krawczyk. SIGMA. http://www.ee.technion.ac.il/ hugo/sigma.html.

[17] D. Kügler. Man in the Middle Attacks on Bluetooth. In *Financial Cryptography '03*, Long Beach, 2003. Lecture Notes in Computer Science, Springer-Verlag.

[18] J.-O. Larsson and M. Jakobsson. SHAKE. Private communication with M. Jakobsson.

[19] D. Maher. United States Patent (No. 5,450,493): Secure communication method and apparatus, 1993.

[20] A. J. Menezes, P. C. van Orschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press LLC, 1997.

[21] T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology–CRYPTO '91*, pages 129–140. Lecture Notes in Computer Science, Springer-Verlag, 1992.

[22] A. Perrig and D. Song. Hash Visualization: A New Technique to Improve Real-World Security. In *Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, pages 131–138, July 1999.

[23] D. Plummer. An Ethernet Address Resolution Protocol, 1982. IETF Standards Track RFC 826.

[24] J.-F. Raymond and A. Stiglic. Security Issues in the Diffie-Hellman Key Agreement Protocol, September 2000. http://citeseer.nj.nec.com/453885.html.

[25] D. Song. dsniff. http://naughty.monkey.org/∼dugsong/dsniff/.

[26] F. Stajano. *Security for Ubiquitous Computing*. John Wiley & Sons, Ltd., 2002.

[27] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proceedings of the 7th International Workshop on Security Protocols*, 1999.

[28] D. S. Wong, H. H. Fuentes, and A. H. Chan. The Performance Measurement of Cryptographic Primitives on Palm Devices. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*.