Attacks on the Pairing Protocol of Bluetooth v2.1

Andrew Y. Lindell*

June 25, 2008

Abstract

The Bluetooth protocol for close-range wireless communication has been a huge success. It is a widely adopted standard and is used for a wide range of devices, from cellphones to PDAs to laptops and more. Due to its ubiquity and importance, its security has become a critical issue. In the new version 2.1 released in July 2007, a complete overhaul of the pairing procedure was carried out with the express aim of making it more secure. In this paper we show that the Bluetooth pairing protocol in passkey entry mode *completely leaks the password*. In addition, we show that it is possible to pair with a device that uses a fixed (but unknown) password, even when the password is random and reasonably long. Our attacks demonstrate that passkey entry mode can only be used with a different random password each time. Unfortunately this is not possible for devices that use a fixed password (like many hands-free car kits). In addition, due to human behavior, this is unlikely to be the case when the user enters the password into two devices in order to pair them. Thus, devices who leave it to the user to enter a password (instead of randomly generating it on one of the devices) will be vulnerable to attack.

1 Introduction

The new version 2.1 of the Bluetooth specification was released recently in July 2007 and contains significant changes across the board, and in particular, to the security infrastructure and pairing procedure. (The pairing procedure is the protocol by which two devices exchange secret keys that they can later use in order to communicate securely.) The designers' stated goal was to protect against eavesdropping and man-in-the-middle attacks; see Section 5.3 of vol 1 of the specification [1]. In order to do this, the pairing protocol is based on Elliptic curve Diffie-Hellman key exchange. This direction is very positive and shows that the security of Bluetooth was considered seriously.

Unfortunately, despite good intentions, we show that the pairing protocol in passkey-entry mode (where a password is used to protect the device) is extraordinarily vulnerable to attack. We present two attacks. In the first we show that an attacker who eavesdrops on a legitimate pairing procedure between two devices can learn the password in under a second; this holds even if the devices share a completely random 64 or 128 bit key! Such an attack does require the attacker to eavesdrop on a pairing procedure, which arguably can be difficult to catch. However, as demonstrated by [8] it is possible to trick devices into rerunning the pairing procedure. Our attack is far more effective than the analogous attack on the pairing protocol of version 2.0, as demonstrated in [8]. Thus, *in this sense*, Bluetooth 2.1 is a step backwards from Bluetooth 2.0 (in other respects the new protocol is an improvement; see below).

*Aladdin Knowledge Systems and Bar-Ilan University, ISRAEL. email: andrew.lindell@aladdin.com, lindell@cs.biu.ac.il.

In addition to the above, we present an active attack that can be carried out on a Bluetooth device. Our attack assumes that the attacker is able to carry out repeated pairing attempts with a device that has a fixed, random password. This is feasible in the case that the device has an automatic pairing feature (something that is unfortunately becoming more and more popular). In addition, it is feasible in the case that the attacker obtains temporary physical access to the device. We stress that *if* such a device has a fixed password comprised of say 6 random digits, then a *secure password protocol* would suffice to prevent pairing (even given physical access or even if automatic pairing is used). Our attack for this scenario has the property that a k-bit password can be learned in approximately $k/2$ interactions with a password-protected device. To be more concrete, a 6 digit password requires approximately 20 bits. In our attack, the equivalent of 10 password guesses suffice to derive the password. Of course, once the password is learned, the attacker can pair with the device. This demonstrates a deep flaw in the pairing protocol of v2.1. We remark that the specification does state that exponentially-increasing delays should be used in the case of failed password guesses. However, since we only require 10 such failures for the common case of a 6 digit password, this will not be effective.

The Bluetooth specification. We stress that the Bluetooth v2.1 specification explicitly refers to a mode whereby a fixed password is used. We quote from Section 3.2.3.1 in vol 3:

> *The PIN may be entered on UI level but may also be stored in the device; e.g. in the case of a device without sufficient MMI for entering and displaying digits.*

Thus, the use of a protocol that only provides protection in the case that a new random password is used in each pairing does not meet the requirements of the specification itself. In order to be fair, we do note that the specification does *consider* a mode of working whereby a new random password be used each time, and generated by the device. We quote from Section 7.7.48 in vol 2:

> *The User Passkey Notification event is used to provide a passkey for the Host to display to the user as required as part of a Simple Pairing process. The Passkey parameter shall be a randomly generated number (see [Part H], Security Specification, Section 2 on page 857) generated by the Controller modulo 1,000,000.*

(Unfortunately, the above way of working is not mandatory; it is possible to implement pairing according to the specification without this. Furthermore, although it seems well understood by the standard's designers, the importance of working in this way is not stressed anywhere in the specification, nor is it stated that if you don't work in this manner then you will be left completely vulnerable.) Thus, another way of interpreting our results is that the above "recommendation" is absolutely mandatory. This would solve the problem regarding devices with interfaces, but still leaves us extraordinarily vulnerable with respect to devices without interfaces that must use a fixed (hopefully random) password.

It is strange, in our opinion, that the Bluetooth SIG chose to ignore the existence of devices without interfaces. As we will see below, the security of such devices can also be critical. The choice of this protocol by Bluetooth is therefore very problematic. As we discuss in Section 5, a very small modification could at least have provided full protection against passive, eavesdropping attackers. This is therefore a serious omission.

Improvements in Bluetooth v2.1. We remark that Bluetooth v2.1 does contain significant improvements over Bluetooth v2.0 in other respects. Specifically, although an eavesdropping adversary can learn the pairing devices' password, it cannot learn the link key and thus the encryption

keys that the parties will use. This is unlike in v2.0 that an eavesdropping adversary can learn both the password and link key, unless the password is very long. Thus, if a new random password is used in every pairing, the protocol of Bluetooth v2.1 provides excellent security. Unfortunately, this cannot always be the case; see below.

Applications of our attacks. As we have mentioned, our attacks are effective when the same password is used to pair multiple times. It is therefore clear that devices like hands-free car kits that do not have user interfaces for entering passwords are vulnerable. This vulnerability exists even for the case of security-conscious manufacturers that hardwire a different random password into every device. Thus, attacks like the car whisperer [4] (where a hands-free car kit is converted into a remote listening device) can be carried out on such devices, even when they migrate to Bluetooth v2.1. Of course, most devices require the user to press a button in order to pair, and seemingly once pairing has been carried out once, there is no reason for the legitimate user to press the pairing button again. However, in reality things are not so simple. First, after the attacker has learned the password it can force the devices to re-pair as demonstrated in [8]. In this re-pairing, the attacker can carry out a man-in-the-middle attack, or can attempt to have its own device pair first. (In this latter case, the user will then have to re-pair yet a third time for its own device. However, a regular user will interpret this state of affairs as the usual hassle that comes with setting up wireless devices.)

Another problem that arises is that more and more devices today are designed with *automatic pairing*. In such a case, if the device has not yet paired, then no button needs to be pressed. However, this opens the door to an attacker to possibly pair *before* the legitimate user (the user will then have to press a button to pair, but is once again likely to interpret this as a malfunction rather than as a breach of security). Furthermore, it is likely that the re-pairing attack of [8] will return the device to a state of not being paired at all, thereby enabling an automatic re-pairing. We may also see devices in the future where automatic pairing is enabled for more than one device. These devices will then be completely vulnerable. One possible argument here is that automatic pairing is inherently insecure and thus "not our problem". However, if the Bluetooth passkey-entry protocol was secure, then it would actually be OK to allow automatic pairing as long as each device is initialized with a unique random password. Thus, such an error could easily be made by manufacturers, even if they are security conscious.

As we have mentioned, the assumption of Bluetooth v2.1 seems to be that passwords will be randomly generated by the devices. This appears in [2] in the following way:

> *The Passkey Entry association model is primarily designed for scenarios where one device has input capability but does not have the capability to display six digits and the other device has output capabilities. A good example of this model is the PC and keyboard scenario. The user is shown a six digit number (from* "000000" *to* "999999"*) on the device with a display. They are then asked to enter the number on the other device. If the value entered on the second device is correct, the pairing is successful.*

The problem is that even though this is the recommended way of working, it is not mandated. Thus, manufacturers – unaware of the dangers – may not work in this way, and may implicitly rely on the assumption that the pairing procedure is secure *as long as the fixed password is random* or *as long as users input good passwords for the pairing.* Let us consider a scenario where two devices (say a PC and PDA) pair by the user inputting a password into both, and let us assume that the user is security-conscious and so uses a random 6 digit number for the password. The question that arises is what will that user do if the pairing fails the first time? When the user has to try again

immediately, then she will most likely enter the same password again. This is due to basic social engineering principles: the user is least likely to interpret a failed pairing as an attack, and will most likely view it as a hardware malfunction. Thus, an attacker can eavesdrop on the first pairing and immediately force a re-pair (using the methodology of [8]). In this case, the user will have to try again right away, opening the door to a man-in-the-middle attack, or as mentioned above enabling the attacker to pair directly with the user's PC or PDA. Note that a successful attack on a pairing between a PDA and a PC (or similar devices) can result in a severe breach of security, and thus the rewards are great enough to motivate such an attack.

The future. It is my strong belief that the Bluetooth SIG should upgrade to a protocol that provides security for devices without interfaces, and that does not leave the way so easily open for mistakes that will leave devices completely vulnerable. More exact recommendations can be found in Section 5.

Related work. Much of the work attacking earlier Bluetooth systems focused on the encryption schemes used; these are not related to our work. The most related works are those that consider attacks on the Bluetooth protocols. For example, see [8] for an attack on the pairing protocol of v2.0, [5] for general security weaknesses, and actual attacks presented in [7, 3, 10]. To the best of our knowledge, this is the first work to consider the security of version 2.1. Another paper that is related is that of [9] who show that these types of protocols are vulnerable when passwords are reused. However, they show weaker results with respect to the Bluetooth protocol.

Organization. In Section 2 we describe the pairing protocol of Bluetooth version 2.1. Then, in Section 3 we present our passive eavesdropping attack and in Section 4 our active attack on password-protected devices. Finally, in Section 5 we present conclusions, suggestions on how to fix the pairing protocol, and recommendations on how to deal with the current situation.

2 The Bluetooth v2.1 Pairing Protocols

There is no doubt that the designers of the pairing protocols in Bluetooth v2.1 had both usability and security in mind. The pairing procedure, arguably the most security-sensitive operation of Bluetooth, has been completely overhauled. In the new specification, there are four modes:

1. *Numeric comparison:* This mode is suitable in the case that both devices have displays that can be used to display a 6 digit number. After running the pairing protocol, the user must verify that both devices display the same number. If yes, the user accepts the pairing as valid. This method of displaying a short number is used to prevent man-in-the-middle attacks on the pairing protocol.

2. *Just works:* This mode runs the same protocol as for numeric comparison, but no comparison is actually made. Pairing run according to this mode is secure against passive eavesdropping attacks, but not against active man-in-the-middle attacks. This can be used together with near-field communication (NFC) to make it harder for an active attack to be carried out. (We remark, however, that wireless communication has a tendency to be amplifiable so one shouldn't trust this too much. Furthermore, even if NFC cannot be amplified, it suffices for someone to brush up against a user's device, and pairing can take place. After this initial pairing, communication continues with regular Bluetooth which has a longer range.)

3. *Out of band:* In this mode, it is assumed that the pairing devices have another channel that can be used to reliably pass information to each other. For example, if one or both of the devices have a keyboard and screen, or if they can be physically connected as well, then these interfaces can be used as an out of band channel. The information passed over the out of band channel verifies that the devices' Bluetooth communication has not been tampered with, and so that a man-in-the-middle attack has not been carried out.

4. *Passkey entry:* In this mode it is assumed that the devices that are pairing both know the same password, and any device not knowing the password should of course fail to pair.

The pairing protocol, called *secure simple pairing*, has 5 stages:

1. *Public-key exchange:* This stage is common to all 4 modes and involves the parties exchanging messages for a basic Diffie-Hellman key exchange over an Elliptic curve group. These messages can be pre-computed and stored in the devices as their public-keys. Thus, this stage involves device A sending its public-key PKa to device B, and device B replying with its public-key PKb. At this point, both parties derive $DHkey$ which is the secret-key derived from the Diffie-Hellman key exchange with messages PKa and PKb.

2. *Authentication stage 1:* This is the stage that is different for all four modes. In actuality, there are only 3 different protocols; the "just works" mode works by running the same protocol as "numeric comparison" but without actually comparing anything. The aim of this stage is for the devices to verify that no man-in-the-middle attack was carried out on the public-key exchange that they ran. We will describe the protocol for the passkey entry mode below.

3. *Authentication stage 2:* The aim of this stage is for the devices to verify that they have successfully concluded the key exchange. This is the same for all three protocols.

4. *Link key calculation:* Once both devices confirm that they have successfully paired, they derive a link key that can be used in the future.

5. *LMP authentication and encryption:* The last step is for the parties to derive keys that are to be used for securing the communication between them.

Our attacks are for the passkey entry mode and are related only to the protocol of *authentication stage 1*. We therefore describe this protocol only and omit details regarding all the other stages (they do not make any difference to the attack and are rather standard).

Authentication stage 1 for the passkey entry mode We use the notation from the Bluetooth specification here. Let r_a denote the password of device A and let r_b denote the password of device B. It is assumed that the user entered the same password into both devices, or that it is displayed on one device and then input into the other (e.g., in the case of a Bluetooth headset for a cellular phone, the headset would come with an embedded password and this would need to be entered by the user into the phone). The protocol is as follows:

1. Let k denote the length of the passwords in bits, and denote the ith bit of ra and rb by rai and rbi, respectively.

2. For $i = 1$ to k, repeat the following:

   (a) Devices A and B choose random nonces Nai and Nbi, respectively. Device A computes the commitment $Cai = f1(PKa, PKb, Nai, rai)$, where $f1$ is HMAC-SHA256 with key Nai and input (PKa, PKb, rai). Likewise, device B computes the commitment $Cbi = f1(PKb, PKa, Nbi, rbi)$.

   (b) Device A sends Cai to device B and device B sends Cbi to device A.

   (c) After receiving Cbi from device B, device A sends the nonce Nai to B.

   (d) Device B checks that
   $$Cai = f1(PKa, PKb, Nai, rbi).$$
   If yes, it sends Nbi to A. If no, it aborts.

   (e) Device A checks that
   $$Cbi = f1(PKb, PKa, Nbi, rai).$$
   If no, it aborts. Otherwise, it proceeds to the next stage.

The Bluetooth specification provides the rationale for this protocol. The basic idea of the protocol is a "gradual release" procedure on the password. Note that device A reveals the bit rai before device B does. However, B is already committed to Cbi and so unless it committed to the correct bit, it will not be able to conclude the protocol without A aborting. This procedure prevents a man-in-the-middle (MITM) attack because the commitments also contain the public keys previously exchanged. Thus, a MITM attacker will have to provide a different commitment value than that sent by the legitimate devices; in particular, it has to provide a commitment value containing its own public-key. However, if it does this, it will have to guess the bits rai, rbi of the password and will thus fail. In addition, and we quote here directly from the specification: "The long nonce is included in the commitment hash to make it difficult to bruteforce even after the protocol has failed". As we will see, such a bruteforce is not needed in order to learn the password bits.

3 An Eavesdropping Attack on the Passkey Entry Mode

In this section, we show that it is possible to trivially learn the devices' password by eavesdropping on a legitimate conversation. As we will show, this attack can be carried out even if the devices' password is very long, and takes under a second.

The attack. Recall that the parties exchange commitments to bits of the passwords in each iteration and they iterate over every bit of the password. An eavesdropping attacker therefore has the following series of values for every i:

$$PKa, PKb, Cai, Cbi, Nai, Nbi.$$

However, given these values the value of rai can be derived by computing a single HMAC-SHA256 computation. Specifically, the attacker simply computes $f1(PKa, PKb, Nai, 0)$. If the result equals Cai it concludes that $rai = 0$; otherwise it concludes that $rai = 1$. We conclude that given the values

$$PKa, PKb, Ca1, Na1, Ca2, Na2 \ldots, Cak, Nak$$

it is possible to derive the full password with only k HMAC-SHA256 computations.

Note that this attack is far easier than carrying out a standard offline dictionary attack, where the attacker obtains pwd and $h(pwd)$ for some hash function. In such an offline dictionary attack, the attacker must traverse the password space and this can be difficult if long and random-looking passwords are chosen. In contrast, even if the devices have a 64 bit random password (something that is admittedly unlikely), an attacker can still learn the password in under a second! This attack is therefore devastatingly simple and requires only the ability of eavesdropping on a legitimate pairing. Note that it is possible to force such a re-pairing and thereby obtain the password; see [8] for details on how this can be achieved.

Preventing MITM attacks. It is one of the express goals of the new pairing protocol to prevent MITM attacks. We remark that in actuality the protocol fails horribly in this goal. This is due to the fact that an attacker needs only eavesdrop on one legitimate pairing in order to learn the devices' password. Following this, it can carry out a MITM attack using the password it has already learned. Using the technique of [8] to force a re-pairing, this means that a MITM attacker can quickly succeed in playing MITM, thereby enabling it to decrypt all traffic between the devices.[1]

Comparison to version 2.0. We conclude that the passkey-entry protocol of the new version 2.1 is *far more vulnerable* to eavesdropping attacks **against the password** than the previous version 2.0 which was vulnerable to a standard offline dictionary attack, as demonstrated in [8]. However, as we have mentioned, an eavesdropping attacker cannot learn the link key between the devices and thus cannot decrypt the communication even if it knows the password. This is due to the fact that the link key is derived from the Diffie-Hellman key exchange and not from the password; the password is used to prevent MITM attacks only. Thus, eavesdropping on a single pairing is not enough (in contrast to version 2.0 where a single eavesdropping sufficed as long as a short password was used).

4 An Active Attack on Password-Protected Devices

In the previous section we described an eavesdropping attack. In order to carry out such an attack, an attacker must eavesdrop on a legitimate pairing conversation between devices (or cause such a pairing to take place). This is of no help if it manages to access a password-protected device but is unable to force the legitimate user to carry out a pairing where it inputs the password. We remark that by a "password-protected device", we mean a device that has a fixed random password; without knowledge of the password it should be impossible to pair even given physical access to the device or even if the device has an automatic pairing feature. In this section, we show that an attacker can learn the secret password and pair with such a device with relative ease.

Let B be a device with a password rb, unknown to the attacker. The attacker plays the initiating device A in a pairing protocol with B as follows:

1. The attacker runs the first steps of the protocol until the authentication stage 1.
2. The attacker computes $Ca1$ using $ra1 = 0$ (alternatively, a random bit could be used each time instead).

[1] We remark that MITM attacks are non-trivial to carry out. However, given that this is an express goal of the protocol designers we find it appropriate to point out. It is our personal opinion that a much greater danger exists due to the fact that once the password is learned, the attacker can obtain direct access to the user's device. In both cases, the protocol fails to protect as it should.

3. After receiving $Cb1$ from device B, the attacker sends $Na1$ to B and observes if B continues or aborts. If B continues, then the attacker knows that $rb1 = 0$. If B aborts, then the attacker knows that $rb1 = 1$. In either case, the attacker knows $rb1$ with full certainty.

4. If B aborted, then the attacker begins from scratch again. However, this time it knows $rb1$ and so can pass the first iteration of the protocol. If B did not abort, then the attacker continues to the second iteration. Thus, in both cases it reaches the second iteration. Then the attacker does the same as before: it computes $Ca2$ using $ra2 = 0$ and determines $rb2$ completely.

5. The attacker continues until it has learned all of the bits $rb1, \ldots, rbk$ and thus has learned the entire password.

Observe that we expect B to abort only about half of the time. Therefore, after $k/2$ interactions with the device, the attacker should learn the entire password. Once again, this means that even devices protected with random long passwords are vulnerable. Furthermore, when a random 6 digit password is used (as would usually be the case), the attacker can learn the password after only 10 attempts.

We remark that the specification states that exponentially-increasing delays should be introduced for repeated failed attempts. However, the number of attempts here is so small, that this is unlikely to be effective. It is instructive to contrast this with the best such attack on a secure password-based authentication protocol; there it would take on average 500,000 attempts to carry out an online attack on a random 6 digit password. This number of attempts can easily be thwarted by increasing delays or a counter limiting the number of failed guesses. However, it is much more difficult in this case.

We conclude this section by remarking that such an attack was not possible on the pairing protocol of version 2.0 and so this is also a step backwards. Indeed, although the protocol of version 2.0 was vulnerable to an offline dictionary attack, it was not vulnerable at all in the case that an attacker has the device but cannot eavesdrop on any legitimate pairing.

5 Conclusions and Suggested Fixes

As we have mentioned, the protocol of version 2.0 of Bluetooth is stronger than this protocol with respect to the above two attacks: an eavesdropper needs to work much harder to find the password (and if the password is long and random the attack fails), and an online guessing attack on the protocol takes much longer requiring a guess of the entire password at once. Despite this, and in order to be fair on the protocol designers, we note that in other respects there are improvements. In particular, an eavesdropper cannot learn the link key between two legitimate devices. Thus, an attacker with *only eavesdropping capabilities* cannot succeed in any attack. The absurdity of this, however, is that in order to protect against such an attacker it would have sufficed to just exchange the public keys PKa and PKb and then have the initiating device send the password encrypted with the derived key $DHkey$ (together with a random challenge from the other device in order to prevent replay attacks). As the protocol designers were clearly aware, such a protocol is not strong enough and thus they constructed a much more complex protocol. Unfortunately, as we have demonstrated, the level of security actually achieved is no better than this simple protocol.

Fixing the protocol. It is possible to prevent the eavesdropping attack that we described with only a minor change to the specification. Specifically, after the public-key exchange phase, the

pairing devices can immediately derive the Diffie-Hellman key $DHkey$. Then, the rest of the protocol – including the crucial authentication stage 1 – can be encrypted under $DHkey$. The result is that an eavesdropping adversary would not learn the password because the Cai, Nai values are encrypted by a key that only the legitimate pairing devices know. This simple fix would therefore prevent the most serious threat that we have uncovered. (We remark that the attacker cannot carry out a man-in-the-middle attack to learn the password, because without first knowing the password such attacks are not possible.) The security model in this case is also clear:

1. If random unique passwords are generated by the device as part of pairing, then full security against active and passive attacks is provided.

2. If a fixed password is used, then security is provided against attacks carried out by adversaries who eavesdrop and try to directly connect with a victim device (but who cannot carry out man-in-the-middle attacks).

As we have mentioned, the above solution does not prevent our active attack described in Section 4. This is due to the fact that the attacker itself knows $DHkey$ because it chose one of the public keys. In order to prevent such an attack, a secure password protocol must be used. Fortunately, such protocols do exist and we recommend the protocol of [6]. This protocol has the cost of approximately twice a Diffie-Hellman key exchange and can be carried out over Elliptic curve groups which, fortunately, have already been implemented. We therefore believe that this is a feasible option. We remark also that the authors of [6] have not patented the protocol. (They do not know if their method is covered by prior patents but since their protocol does not encrypt using the password as a key, it appears that it should not be covered by existing patents.)

Recommendations to end users. What should be done until the passkey-entry protocol is fixed (if it will be fixed)? As far as end users are concerned, the answer is to make sure that you purchase devices that display a random password as part of the pairing procedure. If you cannot purchase such a device, and the type of device you need only comes without an interface, then make sure that it does not have automatic pairing. Beyond this, there isn't very much that can be done (since the device doesn't have interfaces so the user cannot even check that only its devices are paired with it). The only recommendation is therefore to pair in an isolated place, far away from Bluetooth sniffers that may have significant range (unfortunately, a recommendation that is hard to implement in practice).

Recommendation to device manufacturers. As we discussed in the Introduction, the recommendation (or typical scenario) appearing in the Bluetooth specification whereby one device displays a random number which the user then enters to the other should be take to be *mandatory*. We stress that the number is to be chosen uniquely and randomly every time a pairing takes place.

6 Acknowledgements

We would like to thank Joel Linsky for his comments on this work, and for pointing us to the recommendation in [1] for devices with interfaces to generate the password randomly (and not have the user choose it). We would also like to thank Kaisa Nyberg for pointing out the related work in [9].

References

[1] *Specification of the Bluetooth system.* Covered Core Package version 2.1 + EDR. 26 July 2007.

[2] *Simple Pairing White Paper,* 2006-08-03.

[3] Bluejackq. `http://www.bluejackq.com/`, 2004.

[4] CarWhisperer. `http://trifinite.org/trifinite_stuff_carwhisperer.html`, 2005.

[5] M. Jakobsson and S. Wetzel. Security weaknesses in Bluetooth. In *Proc. RSA Security Conf. - Cryptographer's Track*, Springer-Verlag (LNCS 2020), pages 176–191, 2001.

[6] J. Katz, R. Ostrovsky, M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 475–494, 2001. To appear in the *Journal of the ACM.*

[7] Adam Laurie. Serious flaws in Bluetooth security lead to disclosure of personal data. `http://www.bluestumbler.org`, 2003.

[8] Y. Shaked and A. Wool. Cracking the Bluetooth PIN. In *Proc. 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, pages 39–50, 2005.

[9] J. Suomalainen, J. Valkonen and N. Asokan. Security Associations in Personal Networks: A Comparative Analysis. In *ESAS 2007*, Springer-Verlag (LNCS 4572), pages 43–57, 2007.

[10] Ollie Whitehouse. Bluetooth: Red fang, blue fang. *CanSecWest/core04,* April 2004. Available from `http://www.cansecwest.com/csw04/csw04-Whitehouse.pdf`.