Repairing the Bluetooth pairing protocol

Ford-Long Wong, Frank Stajano and Jolyon Clulow

University of Cambridge
Computer Laboratory

Abstract. We implement and demonstrate a passive attack on the Bluetooth authentication protocol used to connect two devices to each other. Using a protocol analyzer and a brute-force attack on the PIN, we recover the link key shared by two devices. With this secret we can then decrypt any encrypted traffic between the devices as well as, potentially, impersonate the devices to each other. We then implement an alternative pairing protocol that is more robust against passive attacks and against active man-in-the-middle attacks. The price of the added security offered by the new protocol is its use of asymmetric cryptography, traditionally considered infeasible on handheld devices. We show that an implementation based on elliptic curves is well within the possibility of a modern handphone and has negligible effects on speed and user experience.

1 Introduction

Bluetooth is an open specification for seamless wireless short-range communications of data and voice between devices. It provides a way to connect and exchange information between wireless-enabled devices such as mobile phones, personal digital assistants (PDAs), laptops, desktops, printers, digital cameras and other peripherals. The specification was first developed by Ericsson, and later formalized by the Bluetooth Special Interest Group.

The Bluetooth authentication protocol is based on symmetric key cryptography and on a (typically numeric and short) password, called PIN, that the user enters into both devices. As first pointed out by Jakobsson and Wetzel [13], the protocol is vulnerable to a passive attack in which the eavesdropper brute-forces the PIN space and silently checks which PIN correctly reproduces the observed message trace.

We implemented this attack and successfully carried it out against several pairs of commercially available Bluetooth devices. Before proceeding any further, let us first review the original Bluetooth authentication and pairing protocol.

1.1 Bluetooth authentication

Bluetooth natively provides authentication and encryption. Authentication is provided by a 128-bit link key, which is a shared secret known to a pair of

B. Christianson et al. (Eds.): Security Protocols 2005, LNCS

Bluetooth devices which have previously formed a pair. The algorithms for authentication and for the generation of link and encryption keys are all based on SAFER+ [22], here used basically as a hash. The cipher algorithm is E0—a stream cipher whose key can be up to 128 bits long.

The security architecture of Bluetooth defines three possible security modes for a device [5,4]. Mode 1 is non-secure, Mode 2 enforces security at the fine-grained service level and Mode 3 enforces security at the link level.

In the case of Modes 2 and 3, if pairing is turned on, when two Bluetooth devices meet for the first time, they pair using the following key:

$$K_{init} = E_{22}\{PIN, BD\_ADDR_A, RAND_B\}$$

where BD_ADDR_A is the device address of device A, $RAND_B$ is a random number contributed by device B and PIN is a shared password that the user must generate and then manually enter into both devices. (Clearly, BD_ADDR_A and $RAND_B$ will be available to an eavesdropper, while PIN won't.)

Once two devices share a link key, the following protocol allows them to renew it and derive a new one, known as a combination key[1], which becomes the new link key used from that point onwards.

The devices each produces a random number (LK_RAND_A or LK_RAND_B), masks it by XORing it with K_{init} and sends it to the other party. Both parties individually hash each of the two random numbers with the Bluetooth address of the device that generated it, using the algorithm E_{21}. The two hashes are then XORed to generate the combination key:

$$\begin{aligned} K_{AB} = {} & E_{21}(LK\_RAND_A, BD\_ADDR_A) \\ & \oplus E_{21}(LK\_RAND_B, BD\_ADDR_B). \end{aligned}$$

The combination link keys calculated by each device after the key agreement should of course be the same if the procedure is successful. The old link key (either K_{init} or a previous K_{AB}) is then discarded.

Another, less secure kind of link key is the unit key K_{unit}, used by devices that don't have the memory to store a link key for each pairing. The restricted memory device will negotiate to use its unit key as the pairwise link key. It will then mask the unit key by XORing it to the K_{init} formed earlier and send it over to the other device.

For authentication, a challenge-response scheme is used. Device A sends B a challenge $RAND_A$, from which Device B must produce the authentication token $SRES$ and transmit it to A, based on the following:

[1] Both the initialization key K_{init} and the combination key are particular types of link key.

$$\{SRES, ACO\} = E_1\{K, RAND_A, BD\_ADDR_B\}$$

The challenge-response is run bilaterally. If encryption is needed, the encryption key is derived from the link key.

2 Breaking Bluetooth

2.1 Guessing the PIN

We used a Bluetooth protocol analyzer [24] to obtain a trace of Bluetooth transmission from the air and to decode the packets. We captured traces of transmissions from commercial Bluetooth devices such as handphones, PDAs, laptops, etc. The packets bearing the required pairing, key formation and authentication processes were analyzed.

We wrote a program that parsed the captured traces and extracted the relevant parameters as they appeared in the transmissions. The trace contains the device addresses, the random numbers exchanged, the challenge and response tokens and all other relevant parameters of one protocol run. Using these, we carried out a kind of dictionary attack (trying first the "easy" PINs such as those made of repeated or sequential digits) and then, where necessary, a full brute force search of the small PIN space[2]. For each PIN we first computed the K_{init}. Then, using the observed intermediate parameters, we computed the combination key and authentication token $SRES$. Both the key agreement using the combination key and the key transport using the unit key can be successfully attacked. If the trace showed that a unit key had been used instead, the number of intermediate parameters is even fewer. Guessing a PIN results in a $SRES$ value identical to the one observed in the trace. The reverse inference holds with high probability.

2.2 Quantitative observations

The attack program was just a prototype and had not been optimised. Running it on a 1.2 GHz Pentium III laptop gave the following timing results (for randomly chosen PINs, of course—the ones subject to pseudo-dictionary attack could be cracked instantaneously). Note that cracking combination keys require more calls to E_{21} compared to cracking unit keys. Even with our non-optimised program, 4-digit PINs can be broken instantaneously, 6-digit PINs take less than a minute and 8-digit PINs can be cracked in less than an hour (Figure 1). This is due to the very small size of the PIN space.

[2] The user interfaces of many Bluetooth devices further restrict the set of characters that may be used in the PIN. For handphones, this set is often just 0 to 9.

Type of key	No. of digits	Time taken
Unit key	4	0.15 s
	6	15 s
	8	25 mins
Combination key	4	0.27 s
	6	25 s
	8	42 mins

Fig. 1. Time to Crack PIN

We have found the unit key used in some models of Bluetooth-equipped PDAs. As will be clear to anyone who knows knows how how the unit key is used, as mentioned in [13] and elsewhere, once the unit key is discovered, the attacker can thereafter freely impersonate the device in its interactions with any other device. Fortunately, we found that at least the unit key was not set to the same string from device to device. The unit key is deprecated from version 1.2 onwards of the Bluetooth specification [6].

The only practical obstacle to widespread replication of the attack described here is that not every would-be eavesdropper will have easy access to a Bluetooth protocol analyzer ($\approx$ 3000 GBP). We expect however that enterprising hackers could in time figure out ways to use cheap standard Bluetooth modules to access the baseband layer directly, without requiring expensive debugging and diagnostic tools.

2.3 Improved Key Management — Frequent Change of Link Key

Although the protocol is broken by an attacker who eavesdrops on the key establishment phase, within its constraints of avoiding public key cryptography it is not an overly weak design. If the attacker misses the initial key establishment, this attack stops working. Moreover, as discussed in Part C, Section 3.4 of Version 1.1 [5] and Vol 2, Section 4.2.3 of Version 1.2 [6] of the specification, Bluetooth already supports renewal of the link key. An attacker who knows the previous link key and eavesdrops on the link key renewal exchange can trivially discover the new key, but if he misses that exchange then the two devices are safe from further eavesdropping, unless they recourse to bootstrapping a new link key from the weak PIN. This is a good security feature but unfortunately it is rarely exploited in current commercial Bluetooth devices. Change of link key is cheap for the user because it does not involve typing a PIN; yet most devices do not offer the option. We propose that devices provide users with the ability to initiate link key change whenever they wish and we further recommend that users exercise this option often. Frequent change of link key forces an attacker to be continually present when the two target devices are communicating. For resistance against an attacker who could be continually present when the link key

is changed, then there is a compelling case to re-key via asymmetric techniques, which is not yet supported in the current specification. Frequent change of link key would also mitigate the risks of Barkan-Biham-Keller-style encryption key replay attacks raised by Ritvanen and Nyberg [27]. Fixes to the Bluetooth cipher algorithm and encryption key derivation are however, beyond the scope of this paper.

After we presented this paper at the workshop, another paper [29] appeared, which performed a similar attack to the one described in Sections 2.1 and 2.2. The computation speed of their attack seemed to be of the same order as ours. An interesting angle in that paper is that the authors attempt to force pairing and repeated pairing instead of passively waiting for one.

2.4 Further comments

Our experiment confirms the expected result that short Bluetooth PINs can be cracked by brute force search with modest computational resources. It is not practical to seek protection in significantly longer PINs. As a rule of thumb, humans on average can reliably remember PINs up to only 7 digits in length, plus or minus 2 [23]. Software optimization and hardware implementations will bring further improvements in attack speed. Computing hardware continues to improve in performance, but the capacity of the human memory does not improve significantly over generations.

One may also observe, although this will have very little practical impact, that even a Bluetooth PIN of the maximal length (16 octets) allowed by the specification will not support the full diversity of 128 bits at the baseband. The PIN is represented at the user interface level by UTF-8 coding—a variable-length character encoding scheme which uses groups of bytes to represent Unicode. This means it is not possible to use the whole theoretical 128 bit PIN space even if the user interface could support it.

The short range of Bluetooth provides some protection against eavesdropping attacks, but an attacker can circumvent this with a high-gain antenna which extends the reach well beyond the Bluetooth nominal range.

A PIN-cracking attack of the kind described in this section could lead, among other threats, to an attacker recording the cleartext of the encrypted communications between your handphone and headset or between your PDA and the desktop machine with which it synchronizes. As a more elaborate follow-up, an active attacker might even be able to impersonate your desktop computer to your PDA and upload maliciously altered data into it, such as an address book with edited phone numbers.

3 Strengthening Bluetooth

After having ascertained the practical vulnerability of the Bluetooth pairing protocol, we did our best to repair it. Our first goal was to establish a strong, eavesdropper-resistant shared key between the two paired devices. We found no way of doing this within the thrifty constraints chosen by the Bluetooth designers, so we ended up having to resort to asymmetric cryptography, in particular to the Diffie-Hellman key agreement. We then sought to make this exchange resistant against active man-in-the-middle attacks. For this we turned to the vast literature on Encrypted Key Exchange [2] and derivatives.

Next, in order to mitigate the overall cost of the algorithm, we implemented the asymmetric cryptography operations using elliptic curves. Finally, to validate the feasibility of the approach, we implemented a single-machine simulation of the whole algorithm (with a single thread performing all the calculations that the two parties would in turn perform during a run of the protocol) and we ported it to a Bluetooth handphone. At the current stage of implementation we are not performing the protocol over radio, because the API of the phone does not allow us to modify the Bluetooth stack, but we are demonstrating that the processor in a modern handphone can perform the protocol with no appreciable penalty in terms of time or energy.

3.1 Password-Based Key Agreement

In the Bluetooth protocol, the eavesdropper may brute-force the PIN offline and learn the session key as we did. To improve on that, we use a variant of Encrypted Key Exchange (EKE). Regardless of the actual PIN, the eavesdropper cannot discover the session key, since it is established via Diffie-Hellman. The PIN instead is used to defeat active middleperson attacks and it cannot be brute-forced because the middleperson is detected at the first wrong guess.

EKE was introduced in 1992 by Bellovin and Meritt [2]. Thereafter, there have been a number of suggestions for password based authenticated key-exchange protocols. These include the schemes described in [11,12,32,7,15]. Some EKE variants add security features such as forward secrecy. Other different approaches include [10], which depend on parties knowing the others' public keys beforehand, [1], which uses collisionful hashes, and [28], which use 'confounders' but neither hashes nor symmetric cryptography.

We presented a protocol at the workshop, which was later discovered to be vulnerable to an active guessing attack when an attacker participates in a protocol run. Details of the protocol and the attack are given in Appendix A.

We thus revise our solution and provide one which is based on the AMP suite developed by Kwon [14,15,16], in particular AMP+. The password is entangled in such a way that an attacker who has no knowledge of the password is not able

to mount such an active guessing attack, and also at the same time not able to compute the shared key formed between two genuine participating parties. The other advantages of this scheme include the following: its relative good efficiency, its simple and understandable structure, it can be easily generalised, ie. to elliptic curve (EC) groups, and a formal security proof is provided.

3.2 Proposed Protocol Implementation

Elliptic Curve Diffie-Hellman - Encrypted Key Exchange (AMP+) The protocol may be sketched as follows. The participants share a weak password P. They hash the password with the parties' identifiers (ie. their device addresses) to produce the password hash s, which is then multiplied with G, the common base point of the elliptic curve, to produce V. Alice sends her ephemeral public key $G.r_A$ [3]. Both Bob and Alice hash their identifiers with Alice's public key to obtain e_1. Bob multiplies Alice's public key by e_1, adds it to V, and multiplies this with Bob's private key r_B. This result is Bob's password-entangled public key, Q_B. Bob sends Q_B to Alice. Both parties would hash both of their identifiers and both parties' public keys together to obtain e_2. Alice computes ω, using her secret key r_A, the password hash s, and the values of e_1 and e_2 computed earlier. After obtaining ω, and using Bob's password-entangled public key Q_B, Alice is able to calculate $(r_A + e_2).r_B.G$ and derive the shared key. Over at Bob's end, he knows r_B, and using Alice's public key Q_A and the value of e_2 he has computed, Bob would likewise be able to calculate $(r_A + e_2).r_B.G$ and derive the shared key. The resulting protocol is shown in Figure 2.

The security of the AMP family as secure authenticated key exchange protocols has been asserted by a formal proof [14] (with the caveats of proofs in this field).

By inspecting the structure of the protocol, it can be seen that a passive eavesdropper would not able to compute the shared session key, unless he knows either r_A or r_B. This is the Diffie-Hellman number-theoretic problem considered hard. An active adversary, Eve, may attempt to carry out a protocol run (either full or partial) so as to obtain a trace to conduct a password guess (either online or offline). She does not have a high chance of success. If Eve attempts to masquerade as Alice, she has one chance at correctly guessing the password hash s, so as to calculate the correct K and subsequently send the correct M_1 to Bob. If she fails at one try, Bob will abort the protocol run without revealing more than one wrong guess. If Eve attempts to masquerade as Bob, and she contributes a password-entangled public key while not knowing the password hash s, even if she manages to collect an M_1 sent by Alice, Eve would need to solve the Diffie-Hellman problem to recover the corresponding r_B that would produce the same K solution which Alice has calculated.

[3] Analogous to the case of multiplicative groups in finite fields, where r_A and g^{r_A} would be the private and public keys respectively, likewise for the EC case, given r_A finding $G.r_A$ is easy, while given $G.r_A$ finding r_A is hard.

#	Alice		Bob
	$s = H_0(I_A, I_B, P)$		$s = H_0(I_A, I_B, P)$
	$V = G.s$		$V = G.s$
	Chooses random r_A		
	$Q_A = G.r_A$		
1		$\xrightarrow{Q_A}$	
	$e_1 = H_1(I_A, I_B, Q_A)$		$e_1 = H_1(I_A, I_B, Q_A)$
			Chooses random r_B
			$Q_B = (Q_A.e_1 + V).r_B$
2		$\xleftarrow{Q_B}$	
	$e_2 = H_2(I_A, I_B, Q_A, Q_B)$		$e_2 = H_2(I_A, I_B, Q_A, Q_B)$
	$\omega = (r_A e_1 + s)^{-1}(r_A + e_2)$		
	$K = H_3(h.Q_B.\omega)$		$K' = H_3(h.(Q_A + G.e_2).r_B)$
	$M_1 = H_4(I_A, I_B, Q_A, Q_B, K)$		
3		$\xrightarrow{M_1}$	
			$M_1' = H_4(I_A, I_B, Q_A, Q_B, K')$
			Verifies $M_1 = M_1'$
			$M_2 = H_5(I_A, I_B, Q_A, Q_B, K')$
4		$\xleftarrow{M_2}$	
	$M_2' = H_5(I_A, I_B, Q_A, Q_B, K)$		
	Verifies $M_2 = M_2'$		

Fig. 2. Password-based Key Agreement (based on AMP+)

Perfect forward secrecy in the protocol is provided through Diffie-Hellman, so an adversary is not able to calculate past session keys after knowing P or s. By the same token, there is also resistance to the Denning-Sacco or known-key attack, in which an adversary may attempt to attack the password after knowing a previous session key.

The protocol resists the two-for-one guessing attack [20] by an adversary which masquerades as Bob. The idea behind this attack is that an attacker can validate two password guesses on one connection attempt. Earlier versions of AMP and SRP [32] were vulnerable to this slight deficiency. The improved AMP version resists this by doing a EC multiply (or exponentiation in discrete logarithm notation) of Q_A by e_1 to obtain Q_B, which breaks the symmetry which would otherwise exist between Q_A and V. Many-to-many guessing attacks, first raised by Kwon [16], particularly affect three-pass protocols. We do not think that many-to-many guessing attacks are too risky for ad-hoc devices though, since it is not expected at present that the devices would participate in more than a single password-based key agreement run at any one time. We however choose to use a four-pass protocol, and not a three-pass one, even if the latter is more efficient, in case of future feature creep where ad-hoc devices become more powerful and get called upon to behave more like server-type machines supporting multiple connection instances.

Key Derivation The co-factor h is used to provide resistance to attacks like small subgroup attacks.

$$K = H_3(h.(r_A + e_2).r_B.G)$$

Further, as recommended by IEEE 1363 Standard Specifications for Public Key Cryptography [25], a key derivation function is used because the output from an elliptic curve secret value derivation step may not always be appropriate to use directly as a session key, due to possible bias in some of its bits. A key derivation function based on some hash function will be able to utilize more effectively the entire entropy of the secret value. The hash also helps resist a known-key attack.

Key Confirmation The key confirmation procedure is necessary to prove that the other party knew the password S and has established the same shared key K. It is bilateral. Including the identifiers of both parties adds explicitness and more defence-in-depth. The key confirmation functions H_4 and H_5 are hash functions. They may be differentiated by having different short strings pre-concatenated with their other hash inputs.

For subsequent mutual authentication between the pair of devices after they have established and confirmed the shared key, they may use the bilateral challenge-response steps similar to what is in Bluetooth's pre-existing specification. We use challenge-response with random nonces then instead of running the earlier key confirmation function again because freshness and resistance to replay attacks would then be necessary.

3.3 Hash Check Values

It has been suggested that checking a short code over a manual transfer channel [9] may be useful to help in key confirmation for wireless device imprinting. In our scheme, these mechanisms can be helpful, but are not essential, since there is already an existing manual data transfer mechanism, the out-of-band sharing of the password, which we assume is performed confidentially, and it is later entangled into the DH keys. The use of short hash check values would present to an adversary the opportunity to search the whole space of 2^r possibilities, where r is the bit length of the short hash, until he finds matching pre-images with which he would be able to carry out a certainly successful middleperson attack. Whereas entangling the password with public keys presents to the adversary a probability of a 1 in 2^t chance to guess correctly in one try the password, where t is the bit length of the password space, to be able to carry out a successful middleperson attack. The lengths of the password and the check value can reasonably be assumed to be of the same approximate order. Under the standard

attacker model of a RF middleperson attacker, hash check values does not appear to be a better alternative.

In scenarios where it is not difficult to present a user-interface which can show check values of sufficient lengths, further using hash check values is recommended, and may even be substituted for the key confirmation step described earlier, to increase the difficulty further for the attacker, in the off-chance that he manages to make a correct guess of the password in one pass. The price is the increased burden for the user of a second "manual transfer" operation.

3.4 Implementation

We developed a demonstration program which implemented the entire key agreement protocol run described above. A 163-bit binary curve is used for the elliptic-curve cryptosystem. The hash function used is SHA-1[4]. Due to the difficulties of integrating this functionality described by the protocol into commercial Bluetooth devices, as changes are required at the baseband level of Bluetooth's protocol stack, we have not proceeded to implement this in a pair of Bluetooth demonstrator devices.

Laptop The unoptimised simulation runs over a 1.2 GHz Pentium M laptop. Our program used routines from the MIRACL [19] library to do the elliptic curve operations. On average, it took 3 milliseconds to perform an elliptic curve multiply operation (the most expensive single operation). As an upper bound, we consider that each of the computations of Q_B and K' requires 2 EC multiplies. The other operations take negligible time with respect to the public-key operations. The entire protocol run is completed in the order of time taken for 6 EC multiplies, which is 18 milliseconds on our platform.

Handphone The software prototype was ported to a commercial handphone, a Nokia 6600 with a 104 MHz ARM processor and running the Symbian operating system. The timing results proved that this phone can run the protocol without any speed problems. An EC multiply of the said order took merely 80 milliseconds. Thus, ignoring communication delays, all the computations of the entire protocol run may be completed in around half a second. If V can be assumed to have been pre-computed, there is a saving of 80 milliseconds. Note that these public key operations, while intensive, are only required for key agreement, which is usually done once-off between a pair of devices. The rest of Bluetooth traffic encryption is carried out by symmetric means.

[4] This choice may be re-evaluated in the light of recent attacks [31], though the usages of the hash functions in the protocol do not appear to require random collision resistance [18].

3.5 Further Work

Having established that asymmetric cryptography is affordable for a modern handphone, there is still the problem of simpler peripherals, such as Bluetooth headsets. It would be useful to develop an "asymmetric protocol", in which the more powerful device perform most of the computation. Another area that would need attention, assuming that the old protocol would be supported alongside the new for compatibility, is that of safeguards against the classic attack of persuading the devices to use the old less secure protocol even when they could both use the new one.

4 Conclusions

It was clear that the Bluetooth pairing protocol, based as it is on symmetric cryptography and on a low entropy PIN, would always be subject to a PIN-cracking attack. We implemented this known attack in order to verify the resources it required. Results indicate that 4-digit PINs (i.e. the ones in the format to which people are most used from their experience with bank cards, car radios and indeed handphones) can be cracked instantaneously. Longer PINs cannot be considered secure, since a non-optimised attack program cracks 8-digit ones in less than an hour. Even longer PINs, and especially PINs in which the characters were more varied than 0–9, would offer somewhat better protection; but we do not believe they are realistic. Within the limits of the existing protocol, frequent change of link key would make the life of the attacker harder. We propose that manufacturers promote the use of this facility.

We have attempted to strengthen the Bluetooth pairing protocol against the attack we could so easily mount. To do so, we have had to resort to asymmetric cryptography. In theory this could be considered as exceeding the design parameters, when taking into account the original design goals and constraints of Bluetooth. Having validated the protocol with a prototype implementation on actual handphone hardware, though, we now suggest that asymmetric cryptography should no longer be axiomatically considered taboo for Bluetooth-class devices. Yes, there will still be a legion of smaller Bluetooth-capable devices with much lesser computational capabilities and energy reserves than our handphone; but it is significant that running a public key protocol on today's mobile phone is much faster than running PGP version 1 or 2 was on a desktop computer of the day, which some of us considered perfectly acceptable.

For the new generation of powerful devices (phones, PDAs, laptops) that are the most likely custodians of digital data that is worth more protection, stronger authentication than that offered by the original Bluetooth protocol would be beneficial, and is now affordable.

Acknowledgements

We are grateful to the members of the Security group at Cambridge, particularly Ross Anderson and George Danezis, for useful comments about earlier versions of the protocol before we attended the workshop.

Appendices

A Attack on early version of protocol

The version of the protocol we presented at the workshop, later found to be insecure, is a variant of the EKE, using Diffie Hellman key exchange over an elliptic curve. We will describe the protocol in the more familiar notation of a multiplicative group over a finite field here. g is the generator of the group, and three hash functions H_0, H_1, and H_2 are used. The shared secret password is P, and its hash, $s = H_0(P)$. The co-factor, h is publicly known. At each instantiation of the protocol run, Alice and Bob choose random ephemeral private values r_A and r_B respectively. C_A and C_B are random challenge nonces. The protocol is shown in multiplicative group form in Figure 3.

#	Alice		Bob
	$Q_A = g^{s+r_B}$		$Q_B = g^{s+r_A}$
1		$\xrightarrow{Q_A}$	
2		$\xleftarrow{Q_B}$	
	$K = H_1((Q_B.g^{-s})^{r_A.h})$		$K' = H_1((Q_A.g^{-s})^{r_B.h})$
3		$\xleftarrow{C_B}$	
4		$\xrightarrow{H_2(I_A,I_B,K,C_B)}$	
			Verify hash
5		$\xrightarrow{C_A}$	
6		$\xleftarrow{H_2(I_B,I_A,K',C_A)}$	
	Verify hash		

Fig. 3. Early version of protocol

A.1 Active guessing attack

While this protocol is secure against a passive offline eavesdropping attack—the class of attack which we had implemented against the pairing protocol in

the existing Bluetooth specification—one of us (Clulow) later found it to be vulnerable to an active guessing attack. An active guessing attack is one in which the attacker does not know the password, but participates in a protocol run with one of the legitimate parties, and uses the trace of the protocol run to calculate the password.

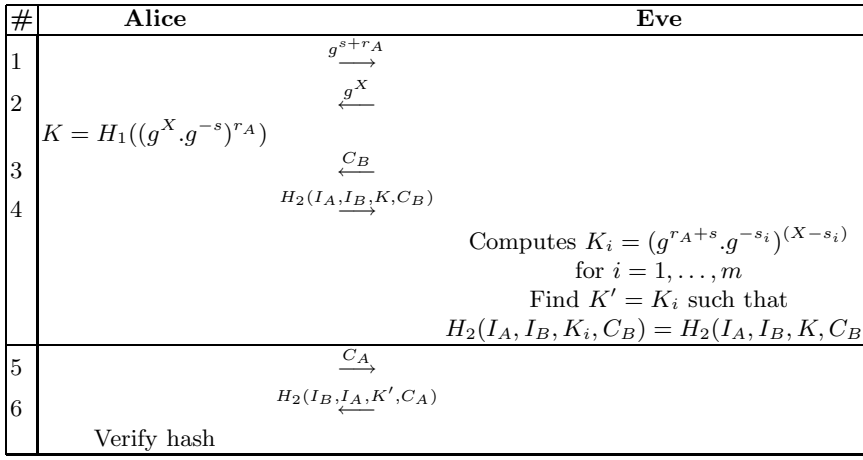

Fig. 4. Active guessing attack on protocol

The vulnerability clearly exists in both the multiplicative group and EC group formulations of the protocol. We will describe it in the former formulation. Let P_i for $i = 1,\ldots,m$ be an enumeration of all possible passwords. We define $s_i = H_0(P_i)$. Eve pretends to be Bob, chooses random X, calculates $Q'_B = g^X$ and submits this value of Alice. Alice continues with the protocol calculating $K = (Q'_B.g^{-s})^{r_A.h}$, and the hash $H_2(I_A, I_B, K, C_B)$ which she sends to Eve. Eve then calculates all possible values of $K_i = (g^{r_A+s}.g^{-s_i})^{(X-s_i).h}$ for $i = 1,\ldots,m$. Eve is also able to calculate all possible values of $H_{2,i} = H_2(I_A, I_B, K_i, C_B)$. She compares the hash of each $H_{2,i}$ to the hash received from Alice. For the value of index i for which $P_i = P$, the hashes will be equal, allowing Eve to identify the original password P with high probability. The basic operations which Eve has to perform are merely exponentiations and hashings, which are computationally tractable. The complete attack is shown in Figure 4.

Depending on the computation power of Eve and the time-out period, she may be able to find the correct K' successfully, and be able to submit Message 6 in time to fool Alice within one protocol run. Alternatively, if Eve is unable to find K' before time-out, she at least has the trace of a failed protocol run to compute K' offline and find P, which she can then use in a subsequent attempt.

Finding an attack which expresses the attacker's view of the key as tractable exponentiation functions, where the inputs are a set of dictionary words, is not

new. What may be considered more novel in this attack against this particular password-based protocol is that the search term s_i for dictionary attack is being applied twice in the expression, ie. $(g^{r_A+s}.g^{-s_i})^{(X-s_i)}$.

B ECDLP-based solution

This appendix contains further details on our elliptic curve implementation of the protocol. A wireless ad-hoc scenario such as Bluetooth gains from having short keys without compromising on cryptographic strength. Storage requirements may be minimized. The computational load on the devices should be minimized, as should the number of messages passed. We can assume a balanced model. In a few applications such for the LAN gateway and the LAN client, the augmented model, similar to that described in [3],which confers resilience to key compromise impersonation, may be useful.

Instead of basing the key agreement on the Discrete Logarithm Problem in a multiplicative group of a prime field (whether it is a traditional discrete logarithm system or a subgroup discrete logarithm system), we may consider it advantageous to base it on the Elliptic Curve Discrete Logarithm Problem (ECDLP).

The algorithms for solving the Discrete Logarithm Problem are classified into index-calculus methods and collision search methods. The latter have exponential running time. Index-calculus methods run at subexponential time, but these require certain arithmetic properties to be present in a group to be successful. The absence of such properties has led to the lack of any known index-calculus attack on elliptic curve discrete logarithms. The best general-purpose algorithm to solve the ECDLP has exponential running time. The current lack of subexponential-time algorithms to solve the ECDLP, as well as the development of efficient implementations of elliptic curve arithmetic, are two main reasons why ECDLP has become attractive on which to base cryptosystems.

B.1 Elliptic Curve Discrete Logarithm

In our proposal, we use a simple construction based on elliptic curves over a binary field. Elliptic curve domain parameters over a binary field are a septuple:

$$T = (m, f(x), a, b, G, n, h)$$

The elliptic curve is specified over a field $GF(q)$ where q is $2m$ for some positive integer m. An irreducible binary polynomial $f(x)$ of degree m specifies the representation of $GF(q)$. Two elliptic curve coefficients a and b, which are elements of $GF(q)$, define an elliptic curve E.

$$E : y^2 + x.y = x^3 + a.x^2 + b$$

A positive prime integer n divides this number of points on E. And G is a base point (x_G, y_G) of order n on the curve. The parameter h is the co-factor. The choice of a binary field is made because this is easier to implement in hardware than the others. Elliptic curves over $GF(2m)$ generally consist of two types of parameters those associated with a Koblitz curve, and those chosen verifiably at random. Koblitz curves allow particularly efficient implementation, but their extra structure also aid attack to some degree. In our trade-off, we prefer randomly chosen parameters.

Based on the recommendations in [17], we may choose an elliptic key size which is equivalent to a symmetric key length of 86 bits (assuming no cryptanalytic progress), or equivalent to a symmetric key length of 79 bits (assuming cryptanalytic effectiveness doubles every 18 months). Such a length is suggested to be 163 bits. The reduction polynomial is:

$$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$$

This is roughly equivalent to 1024 bit length of RSA and DH. Some suggested domain parameters are given in [8], and they are compliant or recommended under ANSI X9.62, ANSI X9.63, IEEE P1363, IPSec and NIST.

References

1. Ross Anderson and Mark Lomas. "Fortifying Key Negotiation Schemes with Poorly Chosen Passwords". *Electronics Letters*, **30**(13):1040–1041, 1994.
2. S. M. Bellovin and M. Meritt. "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks". *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 72–74, May 1992.
3. S. M. Bellovin and M. Meritt. "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise". *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 244–250, 1993.
4. Bluetooth SIG Security Experts Group. *Bluetooth Security White Paper*, **1.0**, April 2002.
5. Bluetooth Special Interest Group. "Bluetooth Specification Volume 1 Part B Baseband Specification". *Specifications of the Bluetooth System*, **1.1**, Feb 2001.
6. Bluetooth Special Interest Group. "Bluetooth Specification Volume 2 Part H Security Specification". *Specification of the Bluetooth System*, **1.2**, Nov 2003.
7. Victor Boyko, Philip Mackenzie and Sarvar Patel. "Provably Secure Password Authentication and Key Exchange Using Diffie-Hellman". *EuroCrypt 2000*, pp. 156–171, 2000.

8. Certicom Corp. *SEC 2: Recommended Elliptic Curve Domain Parameters*, **1.0**, Sep 2000.
9. C. Gehrmann and K. Nyberg. "Enhancements to Bluetooth Baseband Security". *Proceedings of Nordsec 2001*, Nov 2001.
10. L. Gong, M. Lomas, R. Needham and J. Saltzer. "Protecting Poorly Chosen Secrets from Guessing Attacks". *IEEE Journal on Selected Areas in Communications*, **11**(5):648–656, June 1993.
11. David Jablon. "Strong Password-Only Authenticated Key Exchange". *Computer Communication Review*, **26**(5):5–26, Oct 1996.
12. David Jablon. "Extended Password Key Exchange Protocols Immune to Dictionary Attack". *Proceedings of the Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Engineering*, **11**:248–255, June 1997.
13. M. Jakobsson and S. Wetzel. "Security Weaknesses in Bluetooth". *Proceedings of the RSA Conference*, **LNCS 2020**, 2001.
14. Taekyoung Kwon. "Authentication and key agreement via memorable password". *Contribution to the IEEE P1363 study group for Future PKC Standards*, 2000. `http://eprint.iacr.org/2000/026`.
15. Taekyoung Kwon. "Authentication and key agreement via memorable password". *ISOC Network and Distributed System Security Symposium*, Feb 2001.
16. Taekyoung Kwon. "Summary of AMP (Authentication and key agreement via Memorable Passwords)". Aug 2003. `http://dasan.sejong.ac.kr/~tkwon/amp.html`.
17. A. K. Lenstra and E. R. Verheul. "Selecting Cryptographic Key Sizes". *Journal of Cryptology*, **14**:255–293, 2001.
18. Arjen K. Lenstra. "Further progress in hashing cryptanalysis". Feb 2005. `http://cm.bell-labs.com/who/akl/hash.pdf`.
19. Shamus Software Ltd. *Multiprecision Integer and Rational Arithmetic C/C++ Library.* `http://indigo.ie/~mscott/`.
20. Philip MacKenzie. *On the Security of the SPEKE Password-Authenticated Key Agreement Protocol*, July 2001.
21. Philip Mackenzie. "More Efficient Password-Authenticated Key Exchange". *RSA Conference, Cryptographer's Track*, pp. 361–377, 2001.
22. James Massey, Gurgen Khachatrian and Melsik Kuregian. "Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard". *Proceedings of the 1st AES Candidate Conference*, 1998.
23. George A. Miller. "The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information". *Psychological Review*, **63**:81–97, 1956.
24. Mobiwave. *Bluetooth Protocol Analyzer BPA-D10.* `http://www.mobiwave.com`.
25. IEEE P1363. *Standard Specifications For Public-Key Cryptography.* `http://grouper.ieee.org/groups/1363`.
26. S. Patel. "Number Theoretic Attacks on Secure Password Schemes". *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 236–247, 1997.
27. Kaarle Ritvanen and Kaisa Nyberg. "Upgrade of Bluetooth Encryption and Key Replay Attack". *9th Nordic Workshop on Secure-IT Systems*, Nov 2004.
28. M. Roe, B. Christianson and D. Wheeler. "Secure Sessions from Weak Secrets". *Technical report from University of Cambridge and University of Hertfordshire*, 1998.
29. Yaniv Shaked and Avishai Wool. "Cracking the Bluetooth PIN". *to appear in 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, June 2005.

30. Frank Stajano and Ross Anderson. "The Resurrecting Duckling — Security issues for Ad-Hoc Wireless Networks". *Proceedings of the 7th International Workshop on Security Protocols*, 1999.
31. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu. "Finding Collisions in the Full SHA-1". *to appear in Crypto 2005*. `http://www.infosec.sdu.edu.cn/paper/sha1-crypto-auth-new-2-yao.pdf`.
32. Thomas Wu. "The Secure Remote Password Protocol". *Proceedings of 1998 Internet Society Symposium on Network and Distributed System Security*, pp. 97–111, 1998.