Formal Analysis of Authentication in Bluetooth Device Pairing

Richard Chang and Vitaly Shmatikov

The University of Texas at Austin

Abstract. Bluetooth is a popular standard for short-range wireless communications. Bluetooth device pairing enables two mobile devices to authenticate each other and establish a secure wireless connection.
We present a formal analysis of authentication properties of Bluetooth device pairing. Using the ProVerif cryptographic protocol verifier, we first analyze the standard device pairing protocol specified in the Bluetooth Core Specification, which relies on short, low-entropy PINs for authentication. Our analysis confirms a previously known attack guessing attack. We then analyze a recently proposed Simple Pairing protocol. Simple Pairing involves Diffie-Hellman-based key establishment, in which authentication relies on a *human visual channel*: owner(s) of the mobile devices confirm the established keys by manually comparing the respective hash values of the parameters used to generate each key, as displayed on the devices' screens. This form of authentication presents an interesting modeling challenge. We demonstrate how to formalize it in ProVerif. Our analysis shows that authentication can fail when the same device is involved in concurrent Simple Pairing sessions. We discuss the implications of this authentication failure for typical Bluetooth usage scenarios. We then refine our model to incorporate session identifiers, and prove that the authentication properties of Simple Pairing hold in the new model.
Out-of-band human verification based on image- or audio-matching is increasingly used for authentication of mobile devices. This study is the first step towards automated analysis of formal models of human-authenticated protocols.

1 Introduction

Bluetooth radios are becoming ubiquitous in mobile devices such as cell phones, laptops, and even many modern cars. Over one billion Bluetooth-enabled devices have been shipped to date [5]. These devices are often used to store users' private data. For example, a user may enjoy the convenience of being able to wirelessly transfer contact data between his or her laptop and a mobile phone, but probably does not want that contact data to be publicly available to all Bluetooth devices in range. The Bluetooth specification [4] supports the establishment of pairwise symmetric

keys to allow two devices to securely communicate with each other. The process by which a pair of devices establishes the initial symmetric key is called *device pairing*. The device pairing process comprises authentication, generation of the initialization key, and generation of the link key.

Our main contribution is a formal, tool-supported security analysis of two Bluetooth device pairing protocols. The Simple Pairing protocol presents an interesting challenge to formal verification methods because it relies on out-of-band, *human* authentication (explained in more detail below). Human-verifiable protocols based on image- or audio-matching are becoming increasingly popular for mobile device authentication [12, 13, 7]. Although cryptographic security proofs have been manually derived for similar protocols [9, 15], we view our work as the initial step in applying formal methods to this class of protocols.

The first protocol we analyze is the device pairing protocol defined by the Bluetooth Specification [4]. We will refer to this protocol as *standard pairing*. The second is a recently proposed protocol called *Simple Pairing* [6]. Informally, the security properties we verify for both pairing protocols are (1) key secrecy for the initialization key, and (2) authentication of session participants. Intuitively, our key secrecy property states that upon successfully completing a device pairing between devices A and B, the initialization key is known only to A and B. Our authentication property states that, if A has completed a device pairing in which A believes that it has successfully paired with B, then it has indeed paired with B (and vice versa). To support our analysis, we use ProVerif, an automated verifier for cryptographic protocols [3].

As a warm-up exercise, intended to demonstrate the capabilities of formal verification tools as applied to the security analysis of wireless protocols, we use ProVerif to re-discover a known vulnerability in the standard pairing protocol which allows an attacker to impersonate a Bluetooth device after eavesdropping on a successful pairing session. The attack is a guessing attack against the low-entropy, human-memorable shared secret which is used to generate the initialization key [8].

The main part of this paper is devoted to the analysis of the Simple Pairing protocol, which was designed to rectify vulnerabilities caused by the use of low-entropy secrets in the standard pairing protocol. For key establishment, Simple Pairing uses plain, unauthenticated Diffie-Hellman key exchange. Authentication relies on an interesting out-of-band mechanism. Each device computes a short cryptographic hash of the established key and displays it on the device's screen. The two devices' owner(s) visually compares the displayed values and manually confirms that they

match. In other words, authentication is done via key confirmation on a secure human channel, *i.e.*, a human "equality oracle." In this paper, we present the first formal model for this type of authentication amenable to automated analysis. While there are other protocols in the literature that employ similar human authentication mechanisms [12, 13], to the best of our knowledge none of them have been formally analyzed.

When analyzing our model of human key confirmation, ProVerif was unable to prove that the authentication conditions hold. Further manual analysis revealed that the failure is due to an interaction between concurrent protocol sessions. Authentication requires that the keys established by the two participants be equal in any successfully completed protocol session. The hash value displayed by a Bluetooth device does not indicate in *which* session of the protocol the underlying key was established. As our formal analysis shows, even if the human "equality oracle" confirms that two keys (or, more precisely, their hashes) are equal, these keys might not have been established in the same session.

We emphasize that the specification of the Simple Pairing protocol does not appear to allow any input from the user other than a confirmation that two hash values are equal. In this sense, our symbolic model is an appropriate abstraction of user interaction in Simple Pairing. At the same time, the "attack" in the symbolic model is fundamentally concurrent, and only works if the same device is simultaneously engaged in multiple Simple Pairing sessions. Many real Bluetooth implementations can only conduct a single pairing session at any given time—even though this is *not* required by the actual specification, which, by default, appears to permit concurrent executions.

The failure of authentication in our formal model indicates the lack of precision in the Simple Pairing specification. Either the specification should rule out concurrent sessions outright, or it should explicitly require that the hash values presented to the user be accompanied by session identifiers or other means of distinguishing the values established in different sessions. We discuss this further in section 4.3. This may have implications for other wireless protocols, which are secure when multiple instances are executed sequentially, but insecure when they are executed concurrently.

We modify our model to incorporate explicit session identifiers into the equality tests, *i.e.*, in the modified protocol the human oracle is asked to verify not only that the key hashes match, but also that they correspond to the same protocol session. ProVerif has been able to verify both key secrecy and authentication in the modified protocol.

Organization of the paper. Section 2 gives a brief overview of ProVerif. Section 3 describes standard Bluetooth device pairing and our analysis thereof. Section 4 describes the new Simple Pairing protocol based on a human visual channel, and our analysis. Conclusions are in section 5.

2 Analysis Methodology

For our formal verification of Bluetooth device pairing, we use ProVerif, a cryptographic protocol verifier developed by Bruno Blanchet (see [2] for a detailed overview of the syntax and semantics of ProVerif). ProVerif uses a resolution-based solving algorithm which is sound, but not complete. The general approach employed by ProVerif is as follows. The protocol to be verified is specified in a process calculus, explained in more detail below. Each protocol role is represented by a separate process. The complete protocol is modeled by the parallel composition of an unbounded number of copies of the individual protocol role processes.

ProVerif automatically translates these processes into a set of first-order logic formulas (Horn clauses) which abstractly represent the protocol. The solving algorithm takes these clauses as input and determines the set of facts that an attacker can learn from protocol executions. Properties to be verified are modeled as derivability queries. For example, secrecy of a key k can be modeled as a derivability query for the attacker's knowledge k: if ProVerif's solving algorithm terminates and the set of attacker knowledge does not include k, then k has been proved to be secret. Such proofs are sound even with an unbounded number of sessions and without an *a priori* bound on the structural size of messages sent by the attacker.

The process calculus used to specify protocols is an extension of the π-calculus. Among the extensions are function symbols that model cryptographic operations as symbolic "black boxes." Messages are represented by abstract terms. Terms are names, variables, or constructor applications. Constructors are functional symbols used to build terms. For example, encryption is modeled by a constructor: given term m representing a message, and term k representing a symmetric key, the term $\mathit{encrypt}(m, k)$ represents the encryption of m under k generated by the application of the *encrypt* constructor. The process calculus also includes destructors, which are functional symbols that manipulate terms. The decryption destructor, *decrypt*, is modeled by the following reduction rule: $\mathit{decrypt}(\mathit{encrypt}(m, k), k) \rightarrow m$. Other cryptographic primitives are modeled symbolically in a very similar fashion. Communication is modeled by

named "channels," and sending and receiving messages is modeled by inputing and outputting terms over channels.

ProVerif uses the so called Dolev-Yao model of an attacker. The attacker can eavesdrop or intercept any message sent over the network, compute new messages, and send them to protocol participants. In ProVerif's formalism, the attacker's knowledge is represented by a set of terms. Let us call this set A. If a process P sends a term M over channel c, and $c \in A$ (*i.e.*, the channel is public and readable by the attacker, modeled symbolically by the fact that the attacker knows its name), then M is added to A. The attacker can also apply constructors to elements of A, generating new elements for A, and send elements of A over channels in A.

After a protocol has been specified as a set of processes using the process calculus described above, ProVerif automatically translates these processes into a set of Horn clauses. Essentially, these Horn clauses represent the attacker's potential knowledge from protocol executions as a set of implications. A special predicate $attacker(M)$ is used in these clauses to represent the fact that an attacker knows the term M. Another predicate $mess(c, M)$ is used to represent the fact that a message M has been sent by a process over the channel c. The ability of the attacker to receive message M' from the network is represented by the following Horn clause: $mess(c, M') \wedge attacker(c) \Rightarrow attacker(M')$, which basically encodes part of the attacker model above. If a message is sent over a channel that the attacker can read, then the attacker receives that message.

To see how constructors and destructors are translated into Horn clauses, recall the encryption/decryption example above. The *encrypt* constructor is translated into the following clause:

$attacker(m) \wedge attacker(k) \Rightarrow attacker(encrypt(m, k))$.

The *decrypt* destructor is translated into the following clause:

$attacker(decrypt(m, k)) \wedge attacker(k) \Rightarrow attacker(m)$

The solving algorithm employed by ProVerif applies resolution-based theorem proving to this set of clauses to determine the derivability of facts (not unlike logic programming in Prolog). Modeling and verifying basic secrecy properties is relatively straightforward. If one wishes to verify that a private symmetric key, k_s, used in a protocol's specification is secret, the derivation query $attacker(k_s)$ is given as input to the solving algorithm. The solving algorithm applies resolution with free selection to the set of Horn clauses obtained by translating the process calculus specification. If the algorithm terminates and the attacker's knowledge set does not include $attacker(k_s)$, then secrecy of k_s is preserved by the protocol.

ProVerif supports verification of security properties other than basic secrecy. Most of these properties are based on the notion of *equivalence* between processes. For example, consider *strong secrecy*. Intuitively, a value is strongly secret if the attacker cannot observe any difference between a process where this value is used, and an "ideal" process where a completely different value (*e.g.*, a fresh random nonce) is used in the same position. (This definition of secrecy is similar in spirit to cryptographic definitions of secrecy, which are based on real-or-random indistinguishability.) If, in fact, there is an observable difference between the real and ideal processes—for example, some destructor application succeeds for one and fails for the other—then a special "bad" fact is derivable. Therefore, strong secrecy properties are modeled as derivability of this fact. ProVerif also supports verification of weak secrecy (useful for reasoning about low-entropy secrets, as we show in our model for standard pairing) and the so called "correspondence assertions" (used for modeling authentication, and also employed in our models).

While ProVerif's solving algorithm is not guaranteed to terminate, if it does terminate, the resulting proof is valid for an unbounded number of sessions. Also, as our analysis of Simple Pairing shows, failure to prove a protocol secure can be used to identify attack traces associated with potential vulnerabilities.

3 Standard Pairing

This section presents the results of our analysis of the Bluetooth standard pairing protocol. We provide an overview of the protocol, discuss the interesting aspects of formal modeling, and present the results of our ProVerif-based analysis, including the guessing attack found by ProVerif.

3.1 Overview of Standard Pairing

The standard pairing protocol aims to enable a pair of Bluetooth devices, A and B, to generate a symmetric initialization key, K_{init}, mutually authenticate each other, and generate a link key. The link key that is generated during a standard pairing session is derived using the initialization key. The protocol specification provides a number of ways for the link key to be generated. One device's unit key can be used as the link key by sending it to the other device under encryption of the initialization key, or each device can generate a random number, send this number under encryption of the initialization key to the other device, after which

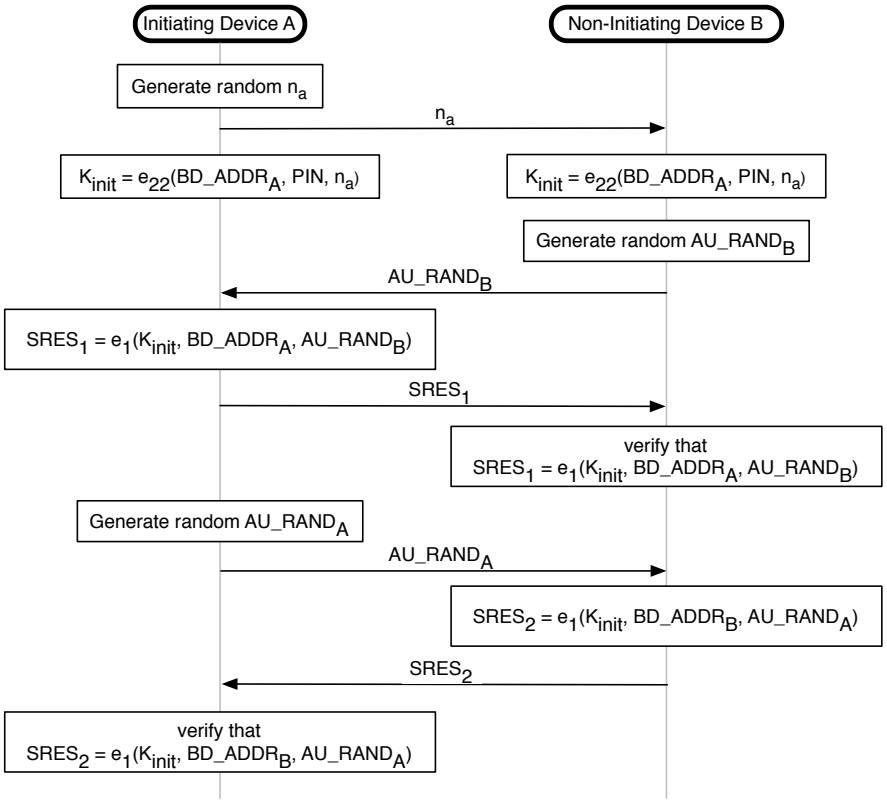

Fig. 1. Sequence diagram for Bluetooth standard pairing

the pair of random numbers are used to generate the link key. In some scenarios introduced in newer versions of the specification the link key is generated before mutual authentication is performed, while in others the link key is generated after mutual authentication is performed using the initialization key. Each of these scenarios suffers from the same problem. An offline guessing attack against the low-entropy secret used to generate the initialization key allows an attacker to impersonate a Bluetooth device. We illustrate this attack using the simplest version, which matches the paper [8] where this attack was first discovered.

The sequence diagram for this protocol appears in Figure 1. Initially, both devices share a low-entropy, human-memorable secret value, PIN. A and B also have access to the each other's Bluetooth device addresses. We call these addresses BD_ADDR_A and BD_ADDR_B for A and B, respectively. The device that initiates the standard pairing session is called

the initiating device. Assume, without loss of generality, that A is the initiating device. B is then the "non-initiating" device (this awkward terminology is borrowed from the Bluetooth product literature).

A begins the protocol by generating a random nonce n_a and sending it to B. Both devices then compute the initialization key K_{init} as a function of n_a, BD_ADDR_A, and PIN (we omit the details of the key generation function, as it has been extensively studied—*e.g.*, see [14, 11, 10]). A and B then execute a two-way challenge-response for authentication. First, A authenticates itself to B. B generates a new random value AU_RAND_B and sends it to A. Upon receiving the new random value, A computes the response as a function of K_{init}, BD_ADDR_A, and AU_RAND_B. A sends this value to B, which verifies the response. The process is repeated with roles reversed to provide mutual authentication.

It is clear from the protocol specification that if an attacker knows the shared PIN value, then both secrecy and authentication will be violated. Key secrecy is violated because all of the parameters used to compute K_{init} are known to the attacker. Additionally, to successfully impersonate a device it is sufficient to know K_{init} and the device's (public) Bluetooth address.

3.2 Analysis of Standard Pairing

As described in section 3.1, breaking secrecy of PIN is sufficient to violate both secrecy and authentication of standard pairing. Therefore, our analysis focuses on secrecy of PIN.

Modeling secrecy of PIN is fairly subtle. In the standard pairing protocol, PIN is intended to be human-memorable, and therefore has low entropy. In practice, 4-digit PINs are used. The main property of low-entropy secrets that we need to model is that they are *guessable*. If an attacker is able to guess a PIN value and verify his or her guess offline, this could constitute a real attack because exhaustively attempting to verify all 4-digit numbers is computationally feasible.

ProVerif supports reasoning about guessing attacks [3]. Modeling this requires declaring term t in question as a *weaksecret*. While executing its solving algorithm, ProVerif attempts to prove an observational equivalence between the attacker's knowledge given a correctly guessed value for t and the attacker's knowledge given a fresh value. Intuitively, should be no observable difference between a process in which the guessed secret is used, and a process in which a different value is used instead. If observational equivalence holds, then there is no guessing attack. In our

ProVerif model for standard pairing, however, observational equivalence fails.

Msg 1. $A \rightarrow B \ : n_a$
Msg 2. $B \rightarrow A \ : AU\_RAND_B$
Msg 3. $A \rightarrow B \ : e_1(e_{22}(BD\_ADDR_A, PIN, n_a), BD\_ADDR_A, AU\_RAND_B)$

Fig. 2. Standard Paring Guessing Attack Messages

The attack found by ProVerif is essentially the same offline guessing attack as described in [8]. Figure 2 shows a transcript of messages that an attacker can use to perform a guessing attack on the shared secret PIN value. An attacker can eavesdrop on a successful standard pairing session between A and B and use these three messages to verify a guessed PIN value offline. Verification works as follows. First, the attacker guesses a value for PIN, say, PIN_g. Then the attacker tries to recompute the third message using PIN_g. In order to compute this message, he or she must know BD_ADDR_A, n_a, and AU_RAND_B. BD_ADDR_A is A's Bluetooth address, which is public. The values of n_a and AU_RAND_B are learned by the attacker from the first and second messages, respectively. To verify whether his guess PIN_g is correct, the attacker uses these values to compute $e_1(e_{22}(BD\_ADDR_A, PIN_g, n_a), BD\_ADDR_A, AU\_RAND_B)$ and compares the result to the third message.

4 Simple Pairing

This section presents of the results of our analysis of the Simple Pairing protocol. We give an overview of the protocol, discuss the authentication scheme used in Simple Pairing, and present the results of our ProVerif-based analysis.

The most interesting aspect of our analysis is our formal model of the "human visual channel," which is essentially a secure equality oracle which enables two Bluetooth devices (or, more precisely, their human owner) to test the established keys for equality. Because equality testing is based on the visual comparison of two numbers, a network attacker cannot interfere with it. Intuitively, a visual channel provides a secure "ideal functionality" for comparing two values for equality. This ideal functionality is then leveraged into complete authentication of the key establishment protocol. Similar mechanisms are used for device authentication in other protocols [12, 13].

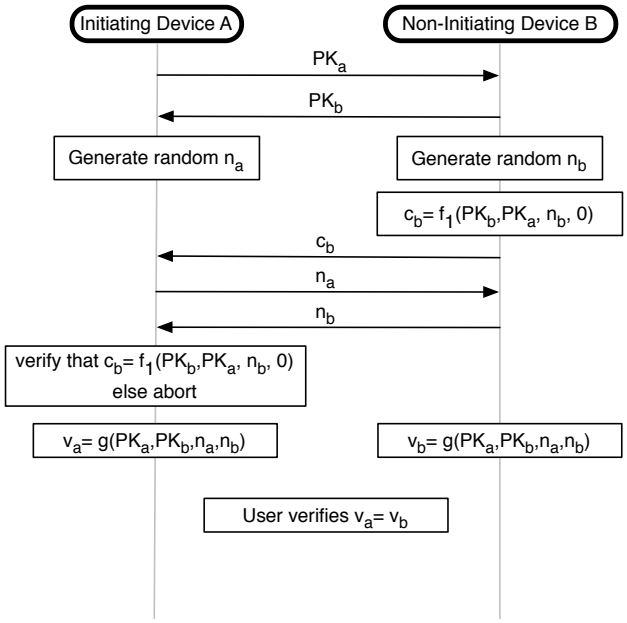

Fig. 3. Sequence diagram for Bluetooth Simple Pairing

4.1 Overview of Simple Pairing

One of the goals of the Simple Pairing protocol is shared with standard pairing: to enable a pair of Bluetooth devices, A and B, to authenticate each other and generate a symmetric initialization key K_{init}. The differences are as follows. First, Simple Pairing has several distinct association models based on the device's hardware capabilities. Second, under most of these association models Simple Pairing does not use any shared secrets at all, let alone low-entropy PINs. The protocol we analyze in this paper uses the *Numeric Comparison* association model [6]. This association model is designed for pairing Bluetooth devices that have displays, such as a cell phone and a laptop. Authentication involves a human oracle: the user is asked to examine the screens of both devices and confirm that they display the same number.

As before, assume that A is the initiating device. First, A and B exchange Elliptic Curve Diffie-Hellman public values PK_A and PK_B. Then A and B generate random values n_a and n_b, respectively. B computes a commitment value c_b which is a collision-resistant function of both Diffie-Hellman values and n_b, and sends c_b to A. A sends n_a to B. B sends

its nonce n_b to A. A re-computes the commitment value c_b and checks whether it is equal to the commitment value previously received from B. Note that there is no authentication so far, *i.e.*, an attacker who controls the communication medium can easily substitute different values into A's and B's messages.

To verify that the parties' views of the conversation match, each device computes a cryptographic hash $H(PK_A, PK_B, n_a, n_b)$ of the two Diffie-Hellman public values PK_A and PK_B, and the two nonces n_a and n_b. The hash value is truncated to a 6 decimal digits, which are displayed on the device's screen. The user is asked to check whether the numbers displayed on the two screens are equal. If the user confirms equality, the devices are considered authenticated, and the symmetric key K_{init} is derived from the joint Diffie-Hellman value and the nonces in the usual way.

4.2 Analysis of Simple Pairing

Our formal model of the Simple Pairing protocol includes a model of the human equality oracle. We formalize it as a special process with two *private channels* which the attacker cannot read or write. Privacy of the channels between the devices and the equality testing process is very important: it models the fact that the user is looking directly at the screen of the Bluetooth device, and the network-based attacker cannot force him to see a number which is different from what the device is displaying. The oracle process reads a value from each channel and outputs a special constant on each channel if the values are equal.

To model secrecy of each device's key, we use standard secrecy in ProVerif, as opposed to weak secrecy. Secrecy of the key values is handled as a derivation query, as mentioned previously.

Modeling authentication is more interesting, and requires *correspondence assertions*. In [1], the ProVerif process calculus is extended to allow processes to emit special `begin` and `end` events that are parameterized by protocol terms. To express authentication of A to B, we modify B's process so that it emits an `end(`B`)` event upon successful completion of a Simple Pairing session, modeling the fact that B has completed a Simple Pairing session. Similarly, A's process is modified to emit a `begin(`B`)` event immediately after public keys are exchanged, modeling the fact that A has started a simple pairing session in which it believes it is pairing with B.

Authentication properties can be expressed as an implication of the following form: if `end(`B`)` is output, then `begin(`B`)` is output. ProVerif supports two types of implication properties, injective and non-injective.

The non-injective version of this property says that if `end(`B`)` is output, then at least one `begin(`B`)` event is output. The injective version is stronger and says that for every `end(`B`)` event, there should be a matching `begin(`B`)` event.

In our analysis, we model both the injective and non-injective versions of two types of authentication properties, which we call weak and strong authentication. *Weak authentication* is modeled by the implication conditions in which the `begin` and `end` events are parameterized only by the participants' identities. If weak authentication holds, then each participant is not mistaken about the identity of the other participant in the protocol, but other session parameters do not necessarily match. *Strong authentication*, on the other hand, is modeled by the implication conditions in which the events are parameterized by all data values used in computing the challenge-response values.

We consider all four types of authentication properties for both A-to-B and B-to-A authentication, modeled modeled as eight correspondence assertion queries. The results of running the ProVerif tool on our Simple Pairing model to check these properties appear in Table 1.

Table 1. Simple Pairing Authentication Properties

Property	Result
A-to-B Injective Weak Authentication	FAILS
B-to-A Injective Weak Authentication	FAILS
A-to-B Non-Injective Weak Authentication	HOLDS
B-to-A Non-Injective Weak Authentication	HOLDS
A-to-B Injective Strong Authentication	FAILS
B-to-A Injective Strong Authentication	FAILS
A-to-B Non-Injective Strong Authentication	FAILS
B-to-A Non-Injective Strong Authentication	FAILS

In our Simple Pairing model, the non-injective weak authentication properties were proved by ProVerif, but all other authentication properties failed, including injective weak authentication and all strong authentication properties. Manually examination of ProVerif's output reveals that the violating derivations involve concurrent execution of multiple Simple Pairing sessions between A and B, and correspond to the situation where the human user's key confirmation is interpreted as successful authentication in a *different* protocol session.

An example of a protocol trace which violates all four strong authentication properties appears in Figure 4. This trace contains messages from two concurrent sessions in which A and B attempt to pair. O represents the equality oracle in our model. Events used by ProVerif to handle correspondence assertions also appear in the trace. They do *not* correspond to actual messages sent in the protocol; we show them to aid the reader in understanding how the properties are violated.

The first part of the trace consists of messages 1 through 7, in which A initiates a Simple Pairing session with B. This session pauses execution immediately after both A and B send their hashes, v_a and v_b respectively, to O. Because this is an uncorrupted session, these values will be the same, and the oracle will eventually return *ok* to both devices. Messages 1' through 7' represent a different session, running concurrently, where again A initiates a Simple Pairing Session with B. In this session, message 4' is actually a corrupted message from an attacker impersonating A, represented by $I(A)$. Instead of receiving A's real nonce for this session, n'_a, B receives n_i. Because A and B's view of A's nonce in the second session differs, the pairing should fail in this session. Specifically, the hashes generated for the human verification step will differ; the keys generated for this session will not agree; and therefore, O should not return *ok* to A and B. Messages 6' and 7' consist of A and B sending their hashes to O for equality verification. At this point in the execution of both sessions, all participants are expecting to receive a reply from O over their respective private channels. Messages 8 and 9 represents a pair of equality confirmation messages to A and B from the first session. In this trace, both instances of A receive message 8 from O and output `end` events corresponding to a successful Simple Pairing session completion. Similarly, both instances of B receive message 9 from O and output the appropriate `end` events. The strong authentication correspondence assertions for the second session are violated by this trace. At the end of this transcript, the event `endBconfirm`(PK_a, PK_b, n_i, n'_b) appears but there is no corresponding `beginBconfirm`(PK_a, PK_b, n_i, n'_b) message. This represents a violation of both the injective and non-injective versions of B-to-A strong authentication. A similar unmatched `endAconfirm`(PK_a, PK_b, n'_a, n'_b) event from the second session appears in the transcript which violates both versions of A-to-B strong authentication.

4.3 Interpreting the Results of Formal Analysis

What do the results of our formal analysis imply about the security of Bluetooth Simple Pairing, as deployed on real-world mobile devices?

Msg 1.	$A \rightarrow B : PK_a$
event beginAparam(PK_a).	
Msg 2.	$B \rightarrow A : PK_b$
event beginBparam(PK_b).	
Msg 3.	$B \rightarrow A : f_1(PK_b, PK_a, n_b, 0)$
Msg 4.	$A \rightarrow B : n_a$
event beginAconfirm(PK_a, PK_b, n_a, n_b).	
Msg 5.	$B \rightarrow A : n_b$
event beginBconfirm(PK_a, PK_b, n_a, n_b).	
Msg 6.	$A \rightarrow O : v_a$
Msg 7.	$B \rightarrow O : v_b$
Msg 1'.	$A \rightarrow B : PK_a$
event beginAparam(PK_a).	
Msg 2'.	$B \rightarrow A : PK_b$
event beginBparam(PK_b).	
Msg 3'.	$B \rightarrow A : f_1(PK_b, PK_a, n'_b, 0)$
Msg 4'.	$I(A) \rightarrow B : n_i$
event beginAconfirm(PK_a, PK_b, n_i, n'_b).	
Msg 5'.	$B \rightarrow A : n'_b$
event beginBconfirm(PK_a, PK_b, n'_a, n'_b).	
Msg 6'.	$A \rightarrow O : v'_a$
Msg 7'.	$B \rightarrow O : v_i$
Msg 8.	$O \rightarrow A : ok$
event endAparam(PK_a).	
event endAparam(PK_a).	
event endAconfirm(PK_a, PK_b, n_a, n_b).	
event endAconfirm(PK_a, PK_b, n'_a, n'_b).	
Msg 9.	$O \rightarrow B : ok$
event endBparam(PK_b).	
event endBparam(PK_b).	
event endBconfirm(PK_a, PK_b, n_a, n_b).	
event endBconfirm(PK_a, PK_b, n_i, n'_b).	

Fig. 4. Simple Paring trace in which strong authentication is violated

Formal verification has been very successful in analyzing (and discovering bugs in) a broad class of network protocols such as SSL/TLS and IKE, which are intended to operate in Internet-like environments, where concurrent execution of multiple instances of the same protocol is a fact of life (*e.g.*, a single Web server may be carrying out hundreds of concurrent TLS sessions). Therefore, protocol analysis tools such as ProVerif are designed to prove that secrecy and authentication hold for an *unbounded* number of concurrent and sequential sessions. In ProVerif, this is modeled by allowing unbounded replication of the authenticating processes.

This strong notion of security is certainly appropriate for Internet security protocols. Bluetooth, on the other hand, is designed for more

constrained environments. In particular, it is not clear whether any Bluetooth device ever has the need to carry out concurrent pairing sessions, or is even capable of doing it. The Simple Pairing white paper does not address concurrent executions at all, nor does it forbid the same device from being engaged in concurrent sessions.

The failure of authentication discovered by ProVerif and illustrated by the trace in Figure 4 represents a genuine attack *when the same device is engaged in concurrent sessions*. In this scenario, the number displayed on the screen for human verification may not have been computed in the session which the user is trying to authenticate.

Effectively, our analysis shows that if the same device is allowed to execute multiple concurrent instances of Simple Pairing, then its user interaction component must associate session identifiers with the hash values it displays to the user (and thus deviate from the Simple Pairing specification). If concurrent executions are not permitted, then this restriction should be clearly spelled out in the protocol specification. Otherwise, Simple Pairing or a similar protocol may end up being deployed in a setting where concurrent executions are a possibility, *e.g.*, when the same mobile phone is trying to pair simultaneously with a music player and a laptop.

One may argue that no "reasonable" implementation would support concurrent sessions without giving the user a means of distinguishing hash values produced in different sessions. This may or may not be true, since such an implementation would *not* be compliant with the current Simple Pairing specification, in which user interaction is limited to displaying two values and asking whether they are equal. It is hard to guess, let alone analyze formally, under what circumstances a "reasonable" implementor might deviate from the protocol specification. It is better to explicitly include all restrictions in the specification.

Normally user interaction is not considered as part of the specification, but in protocols where authentication relies on *human* input (*e.g.*, all protocols authenticated via short authenticated string), security depends on correctness of user interaction, that is, representing the right state of the right instance of the protocol to the user. Our analysis points out not so much an inherent flaw of Simple Pairing, but how important it is that the user interaction piece of the system match the underlying protocol precisely.

4.4 Fixing the Simple Pairing Protocol

The vulnerability discovered in our model involves the human user's key confirmation from one session being interpreted as successful authentication in a different session. Therefore, we propose a minor modification to the Simple Pairing protocol, in which *session identifiers* are included in the messages to and from the human oracle. These identifiers allow each "instance" of a device in a given session to correctly identify which confirmation messages correspond to its session.

We built a process calculus model for the modified Simple Pairing protocol, in which each instance of the same device (*i.e.*, when the same device is participating in multiple instances of the protocol) is assigned a unique session identifier. We also added a check to ensure that each session instance only accepts human oracle responses associated this instance's session identifier. Therefore, there is no possibility that an instance accepts an oracle response intended for a different instance (recall that the channel between the human oracle and the instance is the device's physical interface, presumed to be secure). In the new model, ProVerif was able to prove both the injective and non-injective versions of weak and strong authentication, even in the case of an unbounded number of concurrent sessions.

5 Conclusions

We have presented an automated, tool-supported analysis of two Bluetooth authentication protocols—one based on low-entropy PINs, the other based on human verification of equality. For the standard protocol, our analysis confirms a previously known guessing attack. For the new Simple Pairing protocol, we discover a potential vulnerability caused concurrent executions of authentication, and show how to fix the problem by adding explicit session identifiers to the protocol.

From the verification perspective, our main contribution is a formal model for a non-standard form of authentication, which is based not on cryptography, but on access to a "human oracle" who visually checks whether the key confirmation values derived by the two devices match. This is an increasingly popular device authentication technique [12, 13]. We expect that our work will serve as the first step towards richer formal models of human authentication.

An interesting topic for future work is to develop a proper cryptographic (rather than symbolic) proof of security for Simple Pairing. In

such a proof, human oracles can be modeled as a secure ideal functionality for equality testing, and the protocol can be proved secure relative to this functionality.

References

1. B. Blanchet. From secrecy to authenticity in security protocols. In *Proc. SAS*, 2002.
2. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. IEEE S&P*, 2004.
3. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *Proc. LICS*, 2005.
4. Bluetooth Special Interest Group. Specification of the Bluetooth system. `http://www.bluetooth.com/NR/rdonlyres/1F6469BA-6AE7-42B6-B5A1-65148B9DB238/840/Core_v210_EDR.zip`, 2004.
5. Bluetooth Special Interest Group. Bluetooth wireless technology surpasses one billion devices. `http://www.bluetooth.com/Bluetooth/Press/SIG/BLUETOOTH_WIRELESS_TECHNOLOGY_SURPASSES_ONE_BILLION_DEVICES.htm`, 2006.
6. Bluetooth Special Interest Group. Simple pairing whitepaper. `http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`, 2006.
7. M. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Urzun. Human-verifiable authentication based on audio. In *Proc. ICDCS*, 2006.
8. M. Jakobsson and S. Wetzel. Security weaknesses in Bluetooth. In *Proc. CT-RSA*, 2001.
9. S. Laur and K. Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Proc. CANS*, 2006.
10. O. Levy and A. Wool. A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. In *Proc. SecureComm*, 2005.
11. Y. Lu and S. Vaudenay. Cryptanalysis of Bluetooth keystream generator two-level E0. In *Proc. ASIACRYPT*, 2004.
12. J. McCune, A. Perrig, and M. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proc. IEEE S&P*, 2005.
13. N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Proc. IEEE S&P*, 2006.
14. J. Vainio. Bluetooth security. `http://www.niksula.cs.hut.fi/~jiitv/bluesec.html`, 2000.
15. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Proc. CRYPTO*, 2005.