Formal Verification of Improved Numeric Comparison Protocol for Secure Simple Paring in Bluetooth Using ProVerif

Kenichi Arai and Toshinobu Kaneko
Department of Electrical Engineering, Faculty of Science and Technology, Tokyo University of Science
2641 Yamazaki, Noda, Chiba, 278-8510 Japan

Abstract— *Recently, research has been conducted on automatic verification of cryptographic security protocols with the formal method. An automatic verifier is very useful because the risk of human error in such complicated protocols can be reduced. In this paper, we introduce our formalization of an improved Numeric Comparison protocol for Secure Simple Pairing in Bluetooth proposed by Yeh et al. and verify its security using ProVerif as an automatic cryptographic protocol verifier. As a result, we show that this improved protocol is subject to attacks. Moreover, we propose countermeasures against these attacks on this improved protocol. Our proposal provides this improved protocol with a higher level of security.*

Keywords: Formal Verification, Security, ProVerif, Bluetooth, Secure Simple Pairing, Improved Numeric Comparison Protocol

1. Introduction

Generally, cryptographic protocols hold some security properties, but it is difficult for a non-security specialist to verify the security of cryptographic protocols because of their complexity. Recently, research has been conducted on automatic verification of security with the formal method. ProVerif[1], [2] is a known successful automatic verifier for cryptographic protocols defined in the formal model (the so-called Dolev–Yao model[3]). It is based on a representation of the protocol by Horn clauses[4] and can verify the security properties of secrecy and authentication. Therefore, many cryptographic protocols have been verified by ProVerif[5], [6], and it has succeeded in determining their security weaknesses. The main objective of using an automatic verifier is to reduce the risk of human error in such complicated protocols.

On the other hand, Bluetooth[7], [8], which is built into many devices, is a wireless communication standard connecting digital devices. The mutual authentication procedure between Bluetooth devices is called "pairing." The pairing protocol is only able to select the protocol that uses a personal identification number (PIN) in Bluetooth Core Specification Version 2.0 + EDR[7] and earlier versions. However, an attacker can obtain the PIN with relative ease because many of these Bluetooth devices use a 4-digit PIN or a fixed PIN of commonly known values.

Secure Simple Pairing (SSP) is a protocol that improves the security weakness of the pairing protocol that uses a PIN. SSP is a new pairing protocol specified in Bluetooth Core Specification Version 2.1 + EDR[8]. It uses four models: "Numeric Comparison," "Just Works," "Out Of Band," and "Passkey Entry." However, potential attacks against SSP have been identified in recent years. Chang and Shmatikov proposed an attack against the Numeric Comparison protocol using ProVerif[5]. Lindell, Phan, and Mingard proposed an attack against the Passkey Entry protocol[9], [10]. Moreover, Nomura and Matsuo proposed a more practical attack on this protocol[11]. Yeh et al. pointed out a security weakness in the Numeric Comparison protocol different from Chang and Shmatikov, and proposed an improved version[12].

In this paper, we introduce our formalization of the improved Numeric Comparison protocol proposed by Yeh et al. and verify its security using ProVerif. This paper also discusses countermeasures against attacks on this improved protocol.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce ProVerif, and in Section 3, we introduce Secure Simple Pairing and the improved Numeric Comparison protocol proposed by Yeh et al. In Sections 4 and 5, we introduce our formalization of the improved Numeric Comparison protocol. In Section 6, we show verification results of executing our formalization of the improved Numeric Comparison protocol on ProVerif. In Section 7, we present attacks against the improved Numeric Comparison protocol derived using ProVerif, and in Section 8, we discuss countermeasures against these attacks. We conclude the study in Section 9.

2. ProVerif

ProVerif is an automatic cryptographic protocol verifier in the formal model (the Dolev–Yao model) and enables the verification of the security of cryptographic protocols under the assumption that the cryptographic primitives are idealized. Since the attacker has complete control of the communication channels, it may read, modify, delete, and inject messages.

In ProVerif, cryptographic protocols are described using the syntax (grammar) of Blanchet's process calculus, based on applied π-calculus[13]. The syntax used in this paper is shown as follows:

M, N ::=	terms
a, b, c, k, m, n, s	names
x, y, z	variables
$(M_1, \ldots, M_k)$	tuple
$h(M_1, \ldots, M_k)$	constructor/destructor application
$M = N$	term equality
$M <> N$	term inequality
$\mathbf{not}(M)$	negation
P, Q ::=	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
new $n : t; P$	name restriction
in$(M, x : t); P$	message input
out$(M, N); P$	message output
if M **then** P **else** Q	conditional
let $x = M$ **in** P **else** Q	term evaluation
$R(M_1, \ldots, M_n)$	macro usage
event $e(M_1, \ldots, M_n); P$	events

Please refer to [2] for more details of this syntax. The cryptographic protocol described using this syntax is automatically translated into a set of Horn clauses by ProVerif. It is also possible to describe the cryptographic protocol using a set of Horn clauses from the start.

A clause is a Horn clause if it contains at most one positive literal and is defined as $F_1 \wedge \ldots \wedge F_n \Rightarrow F$ $(\equiv \neg F_1 \vee \ldots \vee \neg F_n \vee F)$, where $n \geq 0$ and F is the only positive literal. F is also called a fact. A Horn clause $F_1 \wedge \ldots \wedge F_n \Rightarrow F$ means that, if all facts $F_1, \ldots, F_n$ are true, then F is also true. A Horn clause with no hypothesis $\Rightarrow F$ is simply written as F. Here, a fact $F = p(M_1, \ldots, M_n)$ expresses a property of the messages $M_1, \ldots, M_n$. p denotes predicates, and several predicates can be used. The term M also represents messages that are exchanged between the protocol's participants. The main predicate used by the Horn clause representation of protocols is attacker: the fact attacker(M) means "the attacker may have the message M." Actions of the adversary and the protocol participants can be modeled because of this predicate.

A set of Horn clauses obtained by automatic translation is called an initial clause. This is composed of the attacker's computational abilities, its initial knowledge, and the cryptographic protocol itself. ProVerif executes a resolution algorithm using initial clauses and verifies whether a fact in contradiction to the desired security property can be derived. When it can, there is an attack against the desired security property. In this case, ProVerif displays an explanation of the actions that the attacker has to perform to break the desired security property. Conversely, when the fact in contradiction to the desired security property cannot be derived, there is no attack. Please refer to [4], [14] for details of the resolution algorithm.

ProVerif can verify the security properties of secrecy[14] and authentication[15]. The verification of secrecy is the most basic capability in ProVerif. To test secrecy of the term M, ProVerif attempts to verify that the state in which the term M is known to the adversary is unreachable. Authentication means "if Alice thinks she is talking to Bob, then she really is talking to Bob." Authentication can be defined using correspondence assertions. These are used to capture relationships between events that can be expressed in the form "if an event e has been executed, then event e' was previously executed."

3. Secure Simple Pairing

In this section, we briefly review SSP[8], and review the improved Numeric Comparison protocol proposed by Yeh et al.[12].

SSP is a new pairing protocol specified in Bluetooth Core Specification Version 2.1 + EDR and has two security goals: protection against passive eavesdropping and protection against man-in-the-middle attacks. It also aims to exceed the maximum security level provided by the use of a PIN with the pairing algorithm used in Bluetooth Core Specification Version 2.0 + EDR and earlier versions.

There are five phases of SSP. Phases 1,3,4, and 5 are the same for all protocols, whereas Phase 2 is different depending on the protocol used.

Phase 1 (Public Key Exchange) exchanges public keys using the Elliptic Curve Diffie–Hellman (ECDH) protocol, and a shared key between both devices is generated. Devices A and B first generate their own ECDH private–public key pair (sk_A,pk_A) and (sk_B,pk_B), respectively, and then each sends its own public key to the other device. Devices A and B then each compute a shared key $DHKey$ using the other device's public key and its own private key.

Phase 2 (Authentication Stage 1) exchanges authentication parameters used by Phases 3 and 4 and confirm these parameters. Phase 2 has three different protocols: Numeric Comparison, Out-of-Band, and Passkey Entry. Note that the Just Works model uses the Numeric Comparison protocol. These protocols are chosen based on the I/O capabilities of both devices. The Numeric Comparison protocol is designed for scenarios where both devices are capable of displaying a 6-digit number and of having the user enter "yes" or "no." The user is shown a 6-digit number on both displays, and then asked whether the numbers are the same on both devices. If "yes" is entered on both devices, the pairing is successful. An example of this protocol is the cell phone/PC scenario.

Phase 3 (Authentication Stage 2) confirms that both devices have successfully completed the exchange.

Phases 4 (Link Key Calculation) and 5 (LMP Authentication and Encryption) compute a link key and an encryption key, respectively. The link key is used to maintain the pairing. The final phase is the same as the final steps in legacy pairing.

Please refer to [8] for more details of the five phases of SSP.

3.1 Improved Numeric Comparison Protocol

The ECDH protocol (Phase 1 of standard SSP) has a security weakness against man-in-the-middle attacks because the senders of the public keys (pk_A, pk_B) are not authenticated.

A man-in-the-middle attack occurs when a user wants to connect devices A and B but instead of directly connecting them, they unknowingly connect to an attacker device M that masquerades as the intended device.

To prevent this attack, a visual number confirmation is designed in the Numeric Comparison protocol (Phase 2 of standard SSP). However, Nokia Research Center conducted a usability experiment and pointed out the possibility that user error occurs when conducting visual number confirmation[16]. SSP has remained vulnerable to man-in-the-middle attacks because of this user error. Therefore, Yeh et al. proposed an improved Numeric Comparison protocol[12]. This improved protocol is composed of three phases and uses a PIN instead of confirming the displayed numbers. We review this improved protocol as follows (Figure 1):

Phase 1: Public Key Exchange and Authentication.

1. The user inputs a PIN on both devices A (the initiating device) and B (the responding device). Devices A and B then each generate their own ECDH private–public key pair (sk_A,pk_A) and (sk_B,pk_B), respectively.
2. Device A XORs pk_A with the PIN and sends A,$IOcapA$ and its XOR value to device B. Here, $IOcapA$ and A are the I/O capability of A and Bluetooth address of A, respectively.
3. Device B XORs the received $(pk_A \oplus PIN)$ with the PIN entered by the user to obtain pk_A and computes a shared key $DHKey$ to pk_A and its own private key. $DHKey$ is computed as a function P192 of these values. Device B then computes a commitment value C_B to $DHKey$, $IOcapB$, $IOcapA$, B, and A. C_B is computed as a function f1 of these values. Device B XORs its own public key with the PIN, and then sends B, $IOcapB$, its XOR value, and C_B to device A. Here, $IOcapB$ and B are the I/O capability of B and Bluetooth address of B, respectively.
4. Device A XORs the received $(pk_B \oplus PIN)$ with the PIN entered by the user to obtain pk_B and computes $DHKey$ to pk_B and its own private key. Device A then computes C_B and compares its C_B with the received C_B. If this check fails, the protocol is aborted. Device A then computes a commitment value C_A to $DHKey$, $IOcapA$, $IOcapB$, A, and B. C_A is computed as the function f1 of these values. Device A then sends C_A to device B.
5. Device B computes C_A and compares its C_A with the received C_A. If this check fails, the protocol is aborted.

Phase 2: Link Key Calculation.

Devices A and B compute a link key LK to the previously shared key ($DHKey$) and the publicly exchanged data (constant string "$btlk$," A, and B). This link key LK is computed as a hash function f2 of these values.

Phase 3: LMP Authentication and Encryption.

After the link key is computed by Phase 2, devices A and B compute an encryption key K_C to the link key (LK), the random number EN_RAND, and ciphering offset number COF. This encryption key K_C is computed as a hash function E_3 of these values.

See [8] for details of the function P192 and hash functions f1,f2,E_3. Note that the ECDH private–public key pair needs to be generated only once per device and may be computed in advance of pairing. Moreover, devices A and B may, at any time, choose to discard the ECDH private–public key pair and generate a new one, although it is not required to do so. These assumptions are the same as those of the standard SSP protocol.

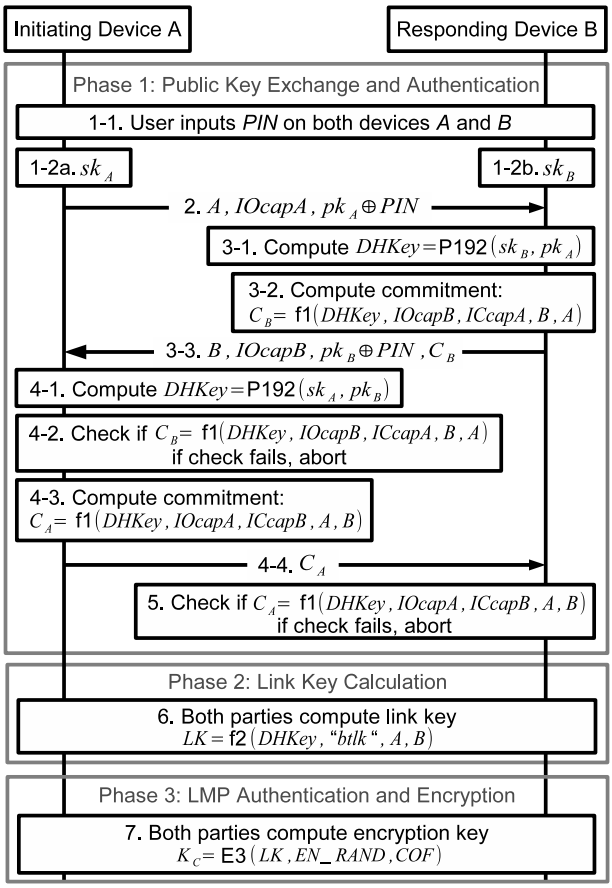

Figure 1: Improved Numeric Comparison protocol

4. Formalization of Cryptographic Primitives

Many cryptographic primitives can be modeled in ProVerif. It is necessary to model function P192, exclusive OR (XOR), hash functions, and symmetric encryption for the formalization of the improved Numeric Comparison protocol. Symmetric encryption and hash functions have already been modeled in ProVerif[2]. In this section, we formalize function P192 and XOR.

4.1 Function P192

Function P192 is defined as follows: given a scalar a and a point P on curve E, the value P192(a,P) is computed as the x-coordinate of the a-th multiple aP of point P. Therefore, function P192 means scalar multiplication on elliptic curves.

We formalize function P192 as follows:

```
1  type G1.
2  type scalar.
3  const P:G1 [data].
3  fun P192(scalar,G1):G1.
4  equation forall a:scalar, b:scalar;
       P192(a,P192(b,P)) = P192(b,P192(a,P)).
```

This formalization is based on the model of the Diffie–Hellman key agreement[2] that has already been formalized and models the ECDH key agreement. This key agreement relies on scalar multiplication in a cyclic group $\mathbb{G}_1$ of prime order q; let P be a point of $\mathbb{G}_1$. Alice selects a random scalar a and sends aP to Bob. Similarly, Bob selects a random scalar b and sends bP to Alice. Alice and Bob then compute $a(bP)$ and $b(aP)$, respectively. These two keys are equal since $a(bP) = b(aP)$ and cannot be obtained by an adversary who has aP and bP but neither a nor b. Here, the elements of $\mathbb{G}_1$ have type G1, the scalars have type scalar, and P is point P. P192 also models scalar multiplication P192(a,P) = aP. The equation at Line 4 means that $a(bP) = b(aP)$.

4.2 Exclusive OR

We formalize XOR as follows:

```
1  fun xor(G1,G1):G1.
2  equation forall x:G1,y:G1; xor(xor(x,y),y) = x.
3  equation forall x:G1; xor(x, xor(x,x)) = x.
4  equation forall x:G1; xor(xor(x,x),x) = x.
5  equation forall x:G1,y:G1; xor(y,xor(x,x)) = y.
```

Here, xor models xor(a,b) = $a \oplus b$. Line 2 means that $((x \oplus y) \oplus y) = x$. Lines 3,4, and 5 refer to the idempotent and associative properties. Note that ProVerif cannot handle the commutative property (that means $(x \oplus y) = (y \oplus x)$) together with the property of Line 2.

5. Improved Numeric Comparison Protocol

5.1 Declarations

The declarations specify a public channel c and cryptographic primitives (constructors/destructors). We formalize declarations as follows (for brevity, we omit cryptographic primitive declarations formalized in Section 4):

```
 1  free c:channel.
 2  free PIN:G1[private].
 3  type tag.
 4  const COF,EN_RAND,btlk:tag.
 5  (* Shared key encryption *)
 6  fun enc(bitstring, G1): bitstring.
 7  reduc forall x:bitstring,y:G1;
      dec(enc(x,y),y) = x.
 8  (* Hash functions *)
 9  type nonce.
10  type key.
11  fun f1(G1,tag,tag,tag,tag):nonce.
12  fun f2(G1,tag,tag,tag):key.
13  fun E3(key,tag,tag):key.
```

Here, we assume that a PIN of the same value is always input to devices A and B. That is, the PIN always uses the same value. Please refer to [2] for details of function/type declarations.

5.2 Security Properties

Authentication can be defined using correspondence assertions[2]. The syntax to query a basic (non-injective) correspondence assertion is **query** x_1:t_1, ..., x_n:t_n; **event**($e(M_1, \ldots, M_j)$) $\Longrightarrow$ **event**($e'(N_1, \ldots, N_k)$). The query is satisfied if for each occurrence of the event $e(M_1, \ldots, M_j)$, there is a previous execution of the event $e'(N_1, \ldots, N_k)$. When the query is not satisfied, the cryptographic protocol of the verification target is subject to an "impersonation attack." The definition of the basic (non-injective) correspondence assertion is also insufficient to capture authentication in cases where a one-to-one relationship between the number of protocol runs performed by each participant is desired. Injective correspondence assertions capture the one-to-one relationship and are denoted as **query** x_1:t_1, ..., x_n:t_n; **inj–event**($e(M_1, \ldots, M_j)$) $\Longrightarrow$ **inj–event**($e'(N_1, \ldots, N_k)$). This correspondence asserts that for each occurrence of the event $e(M_1, \ldots, M_j)$, there is a distinct earlier occurrence of the event $e'(N_1, \ldots, N_k)$. When this query is not satisfied, the cryptographic protocol of the verification target is subject to a "replay attack."

The main objective of SSP is mutual authentication of devices A and B. Accordingly, when device A reaches the end of the protocol with the belief that it has done so with device B, then device B has indeed engaged in a session with device A. The opposite is also true for device B. We declare four events as follows.

event beginAkey(G1,G1), which is used by device B to record the belief that the initiator whose public key and

shared key are supplied as a parameter has commenced a run of the protocol with device B. **event** endAkey(G1,G1), which denotes that device A believes it has successfully completed the protocol with device B. This event is executed only when device A believes it runs the protocol with device B. Device A supplies its public key and shared key $DHKey$ as the parameter. **event** beginBkey(G1,G1), which denotes device A's intention to initiate the protocol with an interlocutor whose device public key and shared key are supplied as a parameter. **event** endBkey(G1,G1), records device B's belief that it has completed the protocol with device A. Device B supplies its public key and shared key $DHKey$ as the parameter.

If device A believes it has completed the protocol with device B, and hence executes the event endAkey, then there should have been an earlier occurrence of the event beginAkey, indicating that device B started a session with device A. Moreover, the relationship should be injective. A similar property should hold for device B.

In addition, we test whether the shared key $DHKey$ is secret at the end of the protocol. The reason for testing the secrecy of $DHKey$ is because the link key and encryption key are computed using $DHKey$. $DHKey$ is a name created by variables such as $DHKeyA$ and $DHKeyB$, while the standard secrecy queries of ProVerif deal with the secrecy of private free names. To solve this problem, the following general technique is used in ProVerif: instead of directly testing the secrecy of the shared keys, ProVerif uses them as session keys to encrypt some free name and test the secrecy of that free name. For example, in the process for device A, we describe enc(secretA,$DHKeyA$) at the end of the protocol and test the secrecy of secretA. SecretA is secret if and only if $DHKeyA$ (that is, the shared key $DHKey$ that device A has) is secret. We proceed symmetrically for device B using secretB.

The ProVerif code to verify the properties of secrecy and authentication can be described as follows:

```
14  (* Secrecy queries *)
15  free secretA,secretB:bitstring[private].
16  query attacker(secretA); attacker(secretB).
17  (* Authentication queries *)
18  event endAkey(G1,G1).
19  event beginAkey(G1,G1).
20  event endBkey(G1,G1).
21  event beginBkey(G1,G1).
22  query x:G1,y:G1; inj-event(endAkey(x,y)) ==>
      inj-event(beginAkey(x,y)).
23  query x:G1,y:G1; inj-event(endBkey(x,y)) ==>
      inj-event(beginBkey(x,y)).
24  (* Secrecy assumptions *)
25  not attacker(new skA).
26  not attacker(new skB).
27
28  (* Device A *)
29  let processA(skA:scalar,pkA:G1,
          A:tag,B:tag,IOcapA:tag,IOcapB:tag) =
30    out(c,(A,IOcapA,xor(pkA,PIN)));
31    in(c,(X:tag,IOcapB':tag,m1:G1,CB1:nonce));
32    let pkX=xor(m1,PIN) in
33    let DHKeyA=P192(skA,pkX) in
34    event beginBkey(pkX,DHKeyA);
35    let CB1'=f1(DHKeyA,IOcapB',IOcapA,X,A) in
36    if CB1=CB1' then
37    let CA1=f1(DHKeyA,IOcapA,IOcapB',A,X) in
38    out(c,CA1);
39    let LKA=f2(DHKeyA,btlk,A,X) in
40    let KCA=E3(LKA,EN_RAND,COF) in
41    event endAkey(pkA,DHKeyA);
42    out(c,enc(secretA,DHKeyA)).
43
44  (* Device B *)
45  let processB(skB:scalar,pkB:G1,
          A:tag,B:tag,IOcapA:tag,IOcapB:tag)=
46    in(c,(Y:tag,IOcapA':tag,m0:G1));
47    let pkY=xor(m0,PIN) in
48    let DHKeyB=P192(skB,pkY) in
49    event beginAkey(pkY,DHKeyB);
50    let CB1=f1(DHKeyB,IOcapB,IOcapA',B,Y) in
51    out(c,(B,IOcapB,xor(pkB,PIN),CB1));
52    in(c,CA1:nonce);
53    let CA1'=f1(DHKeyB,IOcapA',IOcapB,Y,B) in
54    if CA1=CA1' then
55    let LKB=f2(DHKeyB,btlk,Y,B) in
56    let KCB=E3(LKB,EN_RAND,COF) in
57    event endBkey(pkB,DHKeyB);
58    out(c,enc(secretB,DHKeyB)).
59
60  (* Main *)
61  process
62    new skA:scalar; let pkA = P192(skA,P) in
63    new skB:scalar; let pkB = P192(skB,P) in
64    new IOcapA:tag; out(c,IOcapA);
65    new A:tag; out(c,A);
66    new IOcapB:tag; out(c,IOcapB);
67    new B:tag; out(c,B);
68    ((!processA(skA,pkA,A,B,IOcapA,IOcapB)) |
69     (!processB(skB,pkB,A,B,IOcapA,IOcapB)))
```

Queries for secrecy and authentication are specified in Lines 15–16 and Lines 18–23, respectively. Lines 25–26 refer to security assumptions and inform ProVerif that the attacker cannot have the ECDH private key sk_A and sk_B. Process macros for devices A and B are specified in Lines 29–42 and Lines 45–58, respectively. The main process is also specified in Lines 61–69. This process begins by constructing the ECDH private–public key pair (sk_A,pk_A) and (sk_B,pk_B) for devices A and B, respectively. IOcapA,IOcapB,A,and B are then output on the public communication channel c, ensuring they are available to the adversary. An unbounded number of instances of processA and processB are then instantiated with the relevant parameters.

6. Verification Results

Verification results of executing our formalization of the improved Numeric Comparison protocol on ProVerif are shown in Table 1.

This means that the non-injective authentication of device B to A, secrecy for device A (secrecy of secretA), and secrecy for device B (secrecy of secretB) hold; whereas the injective authentications of device A to B and of device B to A, and the non-injective authentication of device A to B are violated.

Property		Result
Secrecy for Device A		True
Secrecy for Device B		True
Injective Authentication	A to B	False
	B to A	False
Non-Injective Authentication	A to B	False
	B to A	True

Table 1: Security properties

The non-injective authentication of device A to B is "false," meaning that device B may end the protocol thinking it has been talking to device A when device A has never run the protocol with device B. This means an impersonation attack. When the injective authentication of device A to B (device B to A) is false, it means that replay attacks are possible for the attacker. If secrecy for device A (device B) is "true," it means that the attacker cannot obtain the shared key $DHKey$. If the non-injective authentication of device B to A is true, it means that impersonation attacks are impossible for the attacker.

7. Derived Attacks

In this section, we review attacks against the improved Numeric Comparison protocol derived using ProVerif.

7.1 Replay Attacks

When the PIN always uses the same value, devices A and B are subject to replay attacks. We explain this attack derived using ProVerif as follows.

Device A sends (A,$IOcapA$,$pk_A \oplus PIN$) and $C_A (= \mathrm{f1}(DHKey, IOcapA, IOcapB, A, B))$ to device B in Phase 1, but these values are always the same. Therefore, an attacker can eavesdrop on communication between both devices during a certain session and obtain these values. The attacker then sends these values to device B, causing a replay attack. Device A is similarly compromised.

7.2 Impersonation Attacks

Device B is subject to impersonation attacks. We explain this attack derived using ProVerif as follows.

Device A sends (A,$IOcapA$,$pk_A \oplus PIN$) to device B in Phase 1. An attacker device M intercepts these values, modifies them to (B,$IOcapB$,m_M), and sends (B,$IOcapB$,m_M) to device B. Here, m_M is a random number that device M generated. Device B then computes $DHKey' = \mathrm{P192}(sk_B, xor(m_M, PIN))$ and $C'_B = \mathrm{f1}(DHKey', IOcapB, IOcapB, B, B)$ using these modified values and sends (B,$IOcapB$,$pk_B \oplus PIN$,C'_B) to device A. Device M eavesdrops on the communication and obtains these values. Device A then computes $DHKey$ and C_B and compares its C_B with the received C'_B. This check fails because its C_B is not equal to the received C'_B, and device A aborts the protocol. Therefore, device A is not subject to impersonation attacks.

Meanwhile, device M sends C'_B instead of sending C_A to device B. Device B then computes $C'_A = \mathrm{f1}(DHKey', IOcapB, IOcapB, B, B)$ and compares its C'_A with the received C'_B. However, this check succeeds because its C'_A is equal to the received C'_B. Therefore, device B is subject to impersonation attacks because there is no check (comparison) after Phase 2.

8. Countermeasures against Attacks

In this section, we propose countermeasures against the attacks mentioned in Section 7.

8.1 Countermeasure against Replay Attacks

In Phase 1, device A sends (A,$IOcapA$,$pk_A \oplus PIN$) and C_A to device B, and device B sends (B,$IOcapB$,$pk_B \oplus PIN$,C_B) to device A. However, these values are always the same; because ($pk_A \oplus PIN$) and ($pk_B \oplus PIN$) values are always the same, $DHKey$ value is always the same. That is, C_A and C_B values are also always the same. Therefore, devices A and B are subject to replay attacks.

Since the values sent by devices A and B are always the same, we change them to a different value, changing the computational method of obtaining $DHKey$. We explain this countermeasure as follows.

[Phase 1-2]: Device A first selects a random number N_A. Device A then concatenates its own public key with N_A, XORs its concatenation value ($pk_A||N_A$) with the PIN, and sends (A,$IOcapA$,$(pk_A||N_A) \oplus PIN$) to device B.

[Phase 1-3]: Device B first selects a random number N_B. Device B XORs the received $(pk_A||N_A) \oplus PIN$ with the PIN entered by the user to obtain pk_A,N_A and computes a shared key $DHKey$ to N_A, its own random number, its own private key, and pk_A. $DHKey$ is computed using a hash function f and function P192 as follows:

$$DHKey = \mathrm{f}(N_A, N_B, \mathrm{P192}(sk_B, pk_A)).$$

Here, we define hash function f using hash functions already defined in SSP. Device B also computes a commitment value $C_B = \mathrm{f1}(DHKey, IOcapB, IOcapA, B, A)$. Device B then concatenates its own public key with its own random number, XORs its concatenation value ($pk_B||N_B$) with the PIN, and sends (B,$IOcapB$,$(pk_B||N_B) \oplus PIN$,C_B) to device A.

[Phase 1-4]: Device A XORs the received $(pk_B||N_B) \oplus PIN$ with the PIN entered by the user to obtain pk_B,N_B and computes $DHKey (= \mathrm{f}(N_A, N_B, \mathrm{P192}(sk_A, pk_B)))$. Device A then computes C_B and compares its C_B with the received C_B. Device A then computes a commitment value $C_A = \mathrm{f1}(DHKey, IOcapA, IOcapB, A, B)$ and sends C_A to device B.

In this countermeasure, we add random numbers to the values sent by devices A and B, and can change these values

to a unique value. Moreover, $DHKey$ can be changed to a unique value using hash function f and random numbers (N_A,N_B).

Note that we have formalized with the assumption that the ECDH private–public key pair is generated only once per device. $DHKey$ can be changed to a unique value by generating the ECDH private–public key pair for each pairing. Therefore, devices A and B are not subject to replay attacks by generating the ECDH private–public key pair for each pairing.

8.2 Countermeasure against Impersonation Attacks

In Phase 1-3, in receiving B and $IOcapB$ sent from the attacker, device B is subject to impersonation attacks. Therefore, device B checks whether the Bluetooth address that it has received is its own (B), and similarly checks whether the I/O capability that it received is its own ($IOcapB$). We add the following procedures to Phase 1-3.

[Phase 1-3]: Device B compares the received Bluetooth address with its own. If it is not the same, the protocol is continued, otherwise the protocol is aborted (A1). Device B then compares the received I/O capability with its own. Again, the protocol is continued if the received I/O capability is not equal to its own, otherwise the protocol is aborted (A2).

```
46    in(c,(Y:tag,IOcapA':tag,m0:G1));
A1    if Y <> B then
A2    if IOcapA' <> IOcapB then
47    let pkY=xor(m0,PIN) in
```

8.3 Verification Results after Countermeasures

Verification results of executing our formalization of the proposed countermeasures on ProVerif are shown in Table 2.

Property		Result
Secrecy for Device A		True
Secrecy for Device B		True
Injective Authentication	A to B	True
	B to A	True
Non-Injective Authentication	A to B	True
	B to A	True

Table 2: Security properties after countermeasures

This means that all properties of secrecy and authentication are held. That is, we have succeeded in making replay and impersonation attacks against the improved Numeric Comparison protocol impossible.

9. Conclusion

In this paper, we introduced our formalization of the improved Numeric Comparison protocol for Secure Simple Pairing in Bluetooth proposed by Yeh et al. and verified its security using ProVerif. We also formalized cryptographic primitives needed to formalize this improved protocol. As a result, we succeeded in deriving replay attacks and impersonation attacks against this improved protocol. We also proposed countermeasures against these attacks on the improved protocol, making them impossible. In future, we would like to verify the security of many cryptographic protocols using ProVerif.

References

[1] B.Blanchet(Project leader), "ProVerif: Cryptographic protocol verifier in the formal model," Available at http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[2] B.Blanchet, B.Smyth, and V.Cheval, "ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," Available at http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[3] D.Dolev and A.Yao, "On the Security of Public Key Protocols," IEEE Transactions on Information Theory, Vol.29(2), pp.198–208, 1983. doi: 10.1109/TIT.1983.1056650.

[4] B.Blanchet, "Using Horn Clauses for Analyzing Security Protocols," Formal Models and Techniques for Analyzing Security Protocols, Cryptology and Information Security Series, Vol.5, pp.86–111, 2011. doi: 10.3233/978-1-60750-714-7-86.

[5] R.Chang and V.Shmatikov, "Formal Analysis of Authentication in Bluetooth Device Pairing," In Proc. of LICS/ICALP Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis, 2007. Available at http://www.cs.utexas.edu/~shmat/shmat_fcs07.pdf.

[6] M.Christofi and A.Goujet, "Formal Verification of the mERA-Based eServices with Trusted Third Party Protocol," Information Security and Privacy Research, IFIP Advances in Information and Communication Technology, Vol.376, 2012, pp.299–314. doi: 10.1007/978-3-642-30436-1_25.

[7] Bluetooth SIG, "Bluetooth 2.0 + EDR Core Specification," 2004. Available at https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40560.

[8] Bluetooth SIG, "Bluetooth 2.1 + EDR Core Specification," 2007. Available at https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=241363.

[9] A.Lindell, "Attacks on the Pairing Protocol of Bluetooth v2.1," In Blackhat USA, 2008.

[10] R.Phan and P.Mingard, "Analyzing the Secure Simple Pairing in Bluetooth v4.0," Wireless Personal Communications, Vol.64(4), pp.719–737, 2012. doi: 10.1007/s11277-010-0215-1.

[11] D.Nomura and K.Matsuo, "A Man-in-the-Middle Attack against Secure Simple Pairing in Bluetooth," IPSJ (Information Processing Society of Japan) Journal, Vol.53(9), pp.2225–2233,2012.(in Japanese)

[12] T.Yeh, J.Peng, S.Wang, and J.Hsu, "Securing Bluetooth Communications," International Journal of Network Security,Vol.14(4),PP.229–235,2012.

[13] M.Abadi and C.Fournet, "Mobile Values, New names, and Secure Communication," In Proc. of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp.104–115, 2001. doi: 10.1145/360204.360213.

[14] B.Blanchet, "An Efficient Cryptographic Protocol Verifer Based on Prolog Rules," In 14th IEEE Computer Security Foundations Workshop, pp.82–96, 2001. doi: 10.1109/CSFW.2001.930138.

[15] B.Blanchet, "From Secrecy to Authenticity in Security Protocols," Static Analysis, Lecture Notes in Computer Science Vol.2477, pp.342–359, 2002. doi: 10.1007/3-540-45789-5_25.

[16] E.Uzun, K.Karvonen, and N.Asokan, "Usability Analysis of Secure Pairing Methods," Financial Cryptography and Data Security, Lecture Notes in Computer Science Vol.4886, pp.307–324, 2007. doi: 10.1007/978-3-540-77366-5_29.