Formal Modeling and Analysis of Bluetooth 4.0 Pairing Protocol

David (Wei) Jia
Stanford University
djia@stanford.edu

Richard Hsu
Stanford University
rhsu@cs.stanford.edu

ABSTRACT

Bluetooth is a wireless technology for exchanging data over short distances and is built into many of the devices used in daily life such as smartphones and laptops. Bluetooth allows communication between paird devices. The pairing is done through Bluetooth's pairing protocol known as Secure Simple Pairing (SSP) which has been established since Bluetooth 2.1+. Bluetooth security is important because sensitive and confidential information such as phone conversations, messages, and key strokes on a keyboard are often communicated through a pairing. We employ a formal model checker to verify the security properties of the pairing protocol used in Bluetooth 4.0, the latest iteration of the protocol. Using our formal Murphi model, we demonstrate and confirm previously known attacks on Bluetooth that have not yet been formalized. We then discuss a new attack found by our model as well as its implications to Bluetooth 4.0 security. Finally, we discuss and recommend possible fixes that could be employed to avoid the attacks we have found. Through this formal modeling we expand on the security verification of Bluetooth pairing protocol.

1. INTRODUCTION

Bluetooth is a wireless technology that is ubiquitous in many modern applications. Users rely on it to transfer data between smartphones, laptops, on-board car systems, and headsets. Bluetooth technology also exists in more peripheral items such as printers, mice, keyboards, and speakers. Because Bluetooth serves as an underlying systems for so many communication systems that are used on a daily basis, senstivie and confidential information can often times be passed through the supposedly secure paired channel. For example, a security breach on Bluetooth could potentially lead to eavesdropping on phone conversations and keylogging password entries. Thus, we analyze Bluetooth in this paper through a formal modeling and analysis in hopes of shedding light on its security.

In this paper, we focus our attentions on the Bluetooth 4.0 Protocol [**?**], which is the most recent version of the technology at the time of writing.

2. BLUETOOTH OVERVIEW

Bluetooth is a wireless technology standard used for exchanging data over a short distance. It utilizes short-wavelength radio transmission and is implemented in devices ranging from hands-free headsets to computer peripherals such as a mouse or keyboard. These devices are fairly ubiquitous today. Some examples of the data transferred between devices via Bluetooth include inputs from a wireless keyboard, contacts transferred between phones, files transferred between computers, and even patient health data from medical sensors to servers. Each Bluetooth responder device (such as a laptop) can be paired with up to seven Bluetooth initiator devies (such as a keyboard or mouse).

Because of the wide range in the application of Bluetooth, the security of the Bluetooth communication is vitally important. The major part of Bluetooth that becomes vulnerable to attackers is the the pairing protocol when two or more devices first initiate connections with one another. The connection process creates a personal area network and requires authentication and encryption to remain secure. Since Bluetooth 2.1, Secure Simple Pairing (SSP) has been implemented and further improved in subsequent versions leading to the current version of Bluetooth 4.0 discussed in the current paper.

2.1 Secure Simple Pairing Protocol

Bluetooth 4.0 Secure Simple Pairing contains four main association models depending on the devices involved:

1. Just Works (JW) When at least one device does not have display nor input channel. Example: wireless headset.
2. Numeric Comparison (NC) When both devices have a display and at least one has a binary input channel for a yes or no response. Example: pairing between smartphones.
3. Passkey Entry (PE) When one device has an input channel but no display and the other device has a display but no input channel. Example: Wireless keyboard.
4. Out of Band (OoB) When both devices support a common additional wireless or wired communication technology for purpose of device discovery or cryptographic channels. Example: Near Field Communication (NFC).

The SSP protocol is completed in five phases. SSP follows the same steps in these five phases for all four association models with the exception of phase 2. The process of pairing through the five phases is described in detail below:

2.1.1 Phase 1: Public Key Exchange

In this phase, Elliptic Curve Diffie-Hellamn (ECDH) key exchange is completed by the initiating (Device A) and non-initiating, or responding device (Device B). Both devices generate a public-private key pair using ECDH. The public key of each device is sent to the other device. Note that according to the official Bluetooth Specifications, this key pair do not need to be generated at each time a pairing occurs. Each device may discard its key pair and generate a new one at any time, but this is not a requirement.

2.1.2 Phase 2: Authentication Stage 1

In this stage, each of the four associated models has a different, but relatively similar authentication process. In each of the associated models, both devices generate a random nonce, N_a and N_b, respectively for Device A and Device B. Device B, the non-initiating device then computes a commitment value $C_b = f_1(PK_b, PK_a, N_b, 0)$, where PK_a is the public key for Device A and PK_b is the public key for Device B, and f_1 is SHA256 dependent one-way hash-function that generates a 128-bit value with the given input.

After C_b is generated, it along with N_b is sent to Device A. Device A also sends N_a to Device B. Next, Device A computes the same commitment value $C_a = f_1(PK_b, PK_a, N_b, 0)$ and compares it with C_b. If they do not equal, then the pairing process is aborted. If they are equal, then the process proceeds, and diverges for each of the four associated models as follows until the end of Phase 2.

1. Just Works (JW) In this associated model, since one of the devices can neither display nor input values, Phase 2 ends here and moves on to Phase 3.
2. Numeric Comparison (NC) In this associated model, Phase 2 goes on to generate two verification codes to be displayed by the user. Each device x generates a six-digit verification code $V_x = g(PK_a, PK_b, N_a, N_b)$, and displays it to the user. The user can then determine if $V_a = V_b$ and confirm the pairing and moves on to Phase 3.
3. Passkey Entry (PE) In PE, the two devices decide upon a secret value, created by the users of the devices. If we are pairing two devices that both do not have screens, then the values are agreed upon by the user. If one device has a display, then the device generates a random value, which is shown on the device's display to be inputted into the other (input) device. This shared secret is turned into an n-bit number (n depends on the length of the shared secret). Let the i-th bit for device x be denoted as r_{x_i}. Since the secret is shared, the assumption is that all $r_{a_i} = r_{b_i}$ for all i. Then for each i, a commitment value, $C_{x_i} = f_1(PK_b, PK_a, N_{x_i}, r_{x_i})$ is generated for device x. The twenty commitment values are sent to the other device and compared. If any commitment values do not equal, then we abort the process. Otherwise, we move on to Phase 3.
4. Out of Band (OoB) In OoB, after the commitment stage described completes, a shared secret comparison is done similar to the PE mode, except the external communication is complete through an external channel such as NFC.

2.1.3 Phase 3: Authentication Stage 2

Phase 3 confirms that both devices have successfully completed the exchange during pairing. Device A generates exchange codes $E_a = f_3(DHKey, N_a, N_b, r_b, IOcapA, A, B)$ and Device B generates $E_b = f_3(DHKey, N_b, N_a, r_a, IOcapB, B, A)$, where $DHKey$ is the Diffie-Hellman key created in Phase 1. Both are sent to the other device. Each device then computes the exchange code for the other device and verifies that it is the same.

2.1.4 Phase 4: Link Key Calculation

A Link Key (LK) is created by both devices by $LK = f_2(DHkey, N_{master}, N_{slave}, \text{“}btlk\text{”}, BD_{ADDRmaster}, BD_{ADDRslave})$. Since both devices have been authenticated at this stage, the LK generated by both devices should be equal.

2.1.5 Phase 5: LMP Authentication and Encryption

Phase 5 is the actual communication between the paired devices. The Link Manager Protocol (the controller that handles authentication and encryption) utilizes the link key developed in Phase 4 to perform authentication and encryption during the communication between the paired devices. For example if a device wants to verify a paired device it can perform a challenge-response by sending a nonce to the claimant device in which it generates a value and sends back to the verifier who can then verify the identity of the claimant device.

3. RELATED WORKS

Because of Bluetooth's prevalence and the emergence of mobile technology, several related works have been done in an effort to analyze the security of Bluetooth. Most recently, in 2010, Phan and Mingard [**?**] analyzed SSP in Bluetooth 4.0 in “Analyzing the Secure Simple Pairing in Bluetooth v4.0” which hand modeled and analyzed Bluetooth 4.0 and described three Man in the Middle (MitM) attacks for the NC and PE modes. In their 2007 paper, “Formal Analysis of Authentication in Bluetooth Device Pairing”, Chang and Shmatikov [4] modeled Bluetooth 2.0 using ProVerif cryptographic protocol verifier and verified a known guessing attack and discovered a potential vulnerability which produces concurrent execution of authentication. Similarly, Haataja and Toivanen [8] discussed in “Practical man-in-the-middle attacks against bluetooth secure simple pairing” two more MitM attacks on Bluetooth 2.0 handsets and hand-free devices as well as modeled a vulnerability on the OoB mode given that the attacker has visual contact to the victim devices. Other papers are more general and informative, presenting a high-level view of the Bluetooth protocol, or suggesting potential vulnerabilities and risks associated with Bluetooth without providing formal analysis or proof.

While these papers present a good starting point for modeling Bluetooth 4.0, they are insufficient. First, many of the analysis of Bluetooth are either not formal or performed on

outdated versions, such as v2.x. The most up-to-date analysis of Bluetooth 4.0 by Phan and Mingard encapsulated a non-formal analysis of the NC and PE modes without formal modeling with a cryptographical tool such as Murphi.

In our paper, we perform a formal analysis and modeling (with Murphi) of Bluetooth 4.0, which has several improved and modified security features from its predecessors. We first confirm the findings of other papers, such as that of Phan and Mingard, by modeling MitM attacks on Bluetooth in NC and PE modes using Murphi. We then describe a novel attack on the NC model found by our formal Murphi modeling.

4. MURPHI MODEL CHECKER

We use the Murphi Model Checker in order to model the states of the Bluetooth 4.0 pairing protocol. The Murphi Model Checker allows us to model the different states possible in the protocol and be able to model the packets being sent between device parties while modeling an adversary that has certain access rights to these packets depending on the protocol mode and encryption methods of the packets. We begin by modeling two devices being paired and then from there model multiple devices connecting to a single parent device. Bluetooth only allows at most seven devices paired to a single responder device, therefore scaling can be done in a natural way and we can model larger ecosystems of Bluetooth devices given the right computational resources.

5. BLUETOOTH MODELING

In order to formally model the Bluetooth 4.0 Pairing Protocols we utilize the Murphi modeling tool. In Murphi, the variables represent the values being passed throughout the communication of the pairing as well as the hash values being created by the devices. The states are modeled by the state variables representing the actual states during the pairing protocol as described by the Bluetooth 4.0 core specification files. We model transitions between steps of each phase as transitions in our state variables. We model the intruder as being a general active attacker adhering to the Dolev-Yao model [**?**]. In conjunction with the possible state executions, the Murphi model checker will explore all possible states and verify our security invariants for plausible executions. Our overall model consists of roughly 1000 lines of Murphi code. Please note that for conciseness, we refrain from showing all of our code in this paper and instead include the code in separate repository. In this paper, for purposes of understanding our formalization, we will mainly show a high level overview of our model by presenting the records and adversarial model.

5.1 Design Choices

We model our pairing protocol by defining separate models for the different device roles in SSP. The major models are the initiator–the device that begins the pairing protocol, and the responder–the device that with which the initiator is being paired. The message model is utilized to describe the transitions throughout the various phases of the pairing protocol. In general only phases prior to and up until phase IV are modeled using Murphi rulesets because phase V simply involves encryption and the start of message transimission after the Link Key is calculated in phase IV. After phase III, the link key generation can be done without a transition, given all prior information from phases I to III. Phase IV is represented in our model by denoting two paired devices as "having" a link key by the linkKey boolean variable of initiators and responders (as we will discuss below). Thus, there is no loss of generality of our model because phase IV is the last phase in which any network communication occurs and the final execution stage that verifies the pairing is complete.

5.2 Modeling Protocol Entities

5.2.1 Initiators

In the initiator model, there are seven distinct states defined by the enumeration InitiatorStates. Three states, `I_SLEEP`, `I_SENT_KEY`, and `I_PHASEONE_DONE` describe the initiator's interactions in phase I of the Bluetooth SSP protocol. `I_WAIT_ONCE`, `I_NC_VERIF_SET`, and `I_PHASETWO_DONE` model the initiator's interactions in phase II. `I_WAIT_EVALUE` and `I_PAIRED` model the initiator's interactions in phase III and IV.

Each initiator record holds the current state the initiator is in, the responder with which the initiator is trying to be paired, and a set of saved variables passed from the responder to the initiator. Finally, the linkKey boolean variable denotes whether or not the link key was generated.

```
Initiator: record
    state: InitiatorStates;
    responder: AgentId;
    -- recieved public key in phase 1
    responder_pk: AgentId;
    -- recieved rb value if any
    responder_r: AgentId;
    -- recieved nonce in phase 2
    responder_n: AgentId;
    -- received commitment value
    responder_c: CValue;
    -- link key of the pairing generated
    linkKey:      boolean;
    -- verification token generated NC Phase 3
    vValue:       VValue;
end;
```

5.2.2 Responder

In the responder model there are six distinct states defined by the enumeration ResponderStates. The first two states `R_SLEEP` and `R_PHASEONE_DONE` describe the responder's interaction in Phase I of the protocol. `R_WAIT_NONCE`, `R_NC_VERIF_SET`, and `R_PHASETWO_DONE` model the responder's interactions in phase II. Finally `R_PAIRED` models the responder's interactions in phase III and IV.

Each responder record holds the current state of each pairing, which saves information between a pairing agent and itself, and contains the information received over the network such as the nonce and public key. Finally, the linkKey states whether or not the link key was generated.

```
Pairing: record
  state:          ResponderStates;
  initiator:      AgentId;
```

```
  -- recieved public key in phase 1
  initiator_pk:  AgentId;
  -- received random value
  initiator_r:   AgentId;
  -- recieved nonce in phase 2
  initiator_n:   AgentId;
  -- received commitment value
  initiator_c:   CValue;
  -- link key of the pairing
  linkKey:        boolean;
  -- vvalue of the pairing
  vValue:         VValue;
end;

Responder: record
  pairings: multiset[MaxInitiators] of Pairing;
end;
```

5.3 Modeling Messages

There are 4 major message types during the protocol pairing:

1. Public key messages (`M_PublicKey`) where the device sends its public key in the clear.
2. Nonce messagess (`M_Nonce`) where the device sends a random nonce in the clear.
3. Commitment value message (`M_CommitValue`) where the device generates a commitment hash value based off the public keys and nonces used.
4. Exchange verification message (`M_ExchangeVerif`) where the device generates an exchange hash value based on the public keys, nonces, a random value, their IO capabilities, and the device addresses in use.

Given these states, the Message record is defined as follows:

```
Message : record
  -- message type
  mType:      MessageType;
  -- source of message
  source:     AgentId;
  -- intended destination of message
  dest:       AgentId;

  -- nonce from source to dest
  nonce:      AgentId;
  -- public key
  publickey:  AgentId;
  -- commit value (Phase 2)
  cValue:     CValue;
  -- exchange verification value (Phase 3)
  eValue:     EValue;
end;
```

In each message they contain the values exchanged throughout the protocol as described in our overview of Bluetooth. Since the Murphi model checker cannot model hash functions properly, we model it by assuming that the cryptographic hash functions are collision resistent therefore this means that in order to generate the same hash value the inputs must be equal. Thus we model hash values as a record of their inputs and when checking that the hash values are equal our model verifies that all input values are equal as well. For example below we present the commitment value, the exchange verification value, and the verification code value which are records of all their inputs in our model.

```
-- CValue
CValue : record
  -- when created - pk of sender
  pk_send:  AgentId;
  -- when created - pk of receiver
  pk_recv:  AgentId;
  -- when created - nonce of sender
  n_send:   AgentId;
end;

-- EValue
EValue : record
  -- when created - pk of sender
  pk_send:    AgentId;
  -- when created - pk of receiver
  pk_recv:    AgentId;
  -- when created - nonce of sender
  n_send:     AgentId;
  -- when created - nonce of receiver
  n_recv:     AgentId;
  -- when created - random value generated by receiver
  r_recv:     AgentId;
end;

-- Verification
VValue : record
  -- public key of initiator
  pk_initiator:   AgentId;
  -- public key of responder
  pk_responder:   AgentId;
  -- nonce sent by initiator
  n_initiator:    AgentId;
  -- nonce sent by responder
  n_responder:    AgentId;
end;
```

5.4 Modeling Protocol Transitions

Transitions in the states of the initiator and responder are modeled using Murphi's rulesets. Each ruleset includes a preposition, which if satisfied, allows the programmer to define the next state. We model each state transition based on the transitions of the SSP protocol.

5.4.1 Initiator and Responder Transitions

As we can see inf Fig **??**, an initiator in the `I_SLEEP` state starts SSP by sending its public key to an Agent (which can be a responder or an intruder). Another ruleset is created to check for messages sent to each responder, in which a responder will reply with its own public key to the source (which could be the initiator or an intruder). This process continues into phase II, in which we discuss the model for the JW association model. The other association models are similar enough that we will not discuss them to reduce redundancy. Rulesets are created in which the initiator and

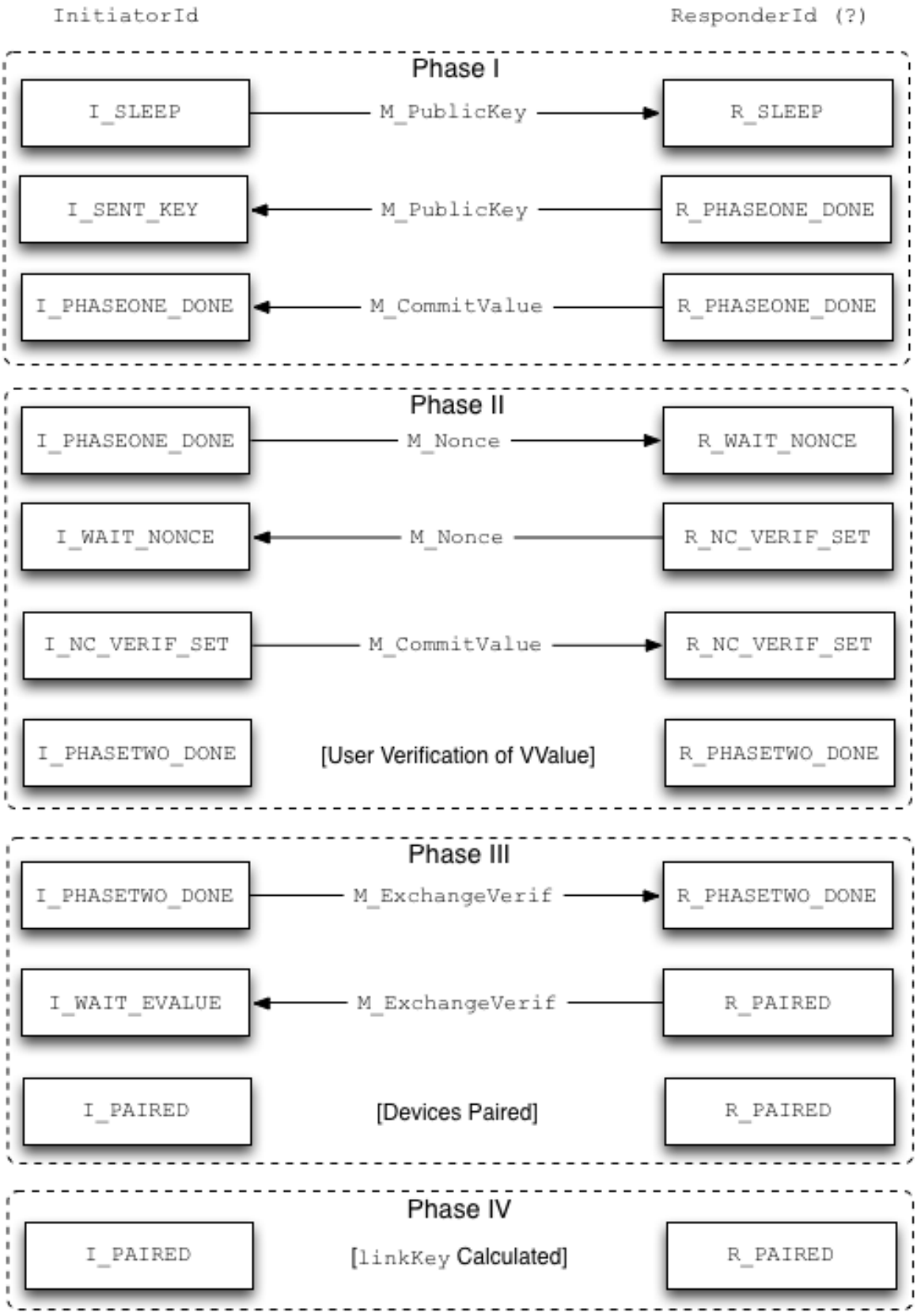

Figure 1: A diagram describing the transitions of the initiator and responder, and the messages that are passed across during SSP. Created by the author.

responder check the network for messages addressed to it that include the nonce and commitment value. These values are checked against received values by the initiator, and if they are confirmed, the protocol continues, otherwise, the message is removed from the network, and the branch of the Murphi search tree stops. We model the checking of the hashed values by checking that all input values are the same, if any are different then the hash would be different. In a similar fashion, phase III is modeled using rulesets for both initiator and responder in which messages with Exchange Verification values are sent across the network and verified by both initiator and responder. If everything verified, we set the initiator state to `I_PAIRED` and the responder state to `R_PAIRED`. Finally the linkKey boolean value is set to true on both the initiator and the responder to represent that the linkKey as been exchanged.t

5.5 Modeling An Adversary

The adversarial model will follow the Dolev-Yao intruder model[**?**] in which the adversary is given the power to intercept, read, store, modify, and forward messages of its choosing. The main adversarial models present against Bluetooth are Man-in-the-Middle (MitM) attacks where an adversary is between the communications of two Bluetooth devices.

5.5.1 Adversary Model

An intruder can intersept messages by taking messages from the network and adding it to its multiset of messages. The intruder record also has an array of linkKeys to represent the initiators and responders of which the intruder has the linkKey. The array of booleans pk and nonce represent whether the intruder has the public key and nonce from a certain agent. The sent_pk and sent_n arra of booleans represent whether the intruder has sent its own public key and nounce to a certain agent.

```
Intruder: record
  messages: multiset[MaxKnowledge] of Message;
  linkKeys: array[AgentId] of boolean;
  -- Do we know their PK
  pk:       array[AgentId] of boolean;
  -- Did we send our PK to them
  sent_pk:  array[AgentId] of boolean;
  -- Do we know their nonce
  nonce:    array[AgentId] of boolean;
  -- Did we send our nonce
  sent_n:   array[AgentId] of boolean;
end;
```

5.5.2 Intercept Messages

The adversarial model against our pairing protocol is mainly a Man-in-the-Middle (MitM) attack in which the adversary has control of the communication channel between the two users. In this case the intruder has three main courses of action: intercept messages, pick a stored message and forward it along without modification, or modify the messages before forwarding it.

```
-- intruder i intercepts messages
ruleset i: IntruderId do
  choose k: net do
    rule 100 ‘‘intruder intercepts messages’’

      -- Pick a msg from network that is not
      -- intended for the intruder
      !ismember (net[k].source, IntruderId)

    ==>

    begin
      -- the message to intercept
      alias msg: net[k] do
        -- If we haven't seen the message, then add
        -- to intruder's messages
        if multisetcount(l:messages,
             int[i].messages[l] = msg
           ) = 0 then
           multisetadd (msg, int[i].messages);
        end;
      end;
      multisetremove (k, net);
    end;
  end;
end;
```

5.5.3 Forward without Modifications

In this model, if an intruder sees a message on the network that has never been seen, then the intruder records the message. The intruder later forwards this message to any possible agent, which is modeled through different branches of Murphi based on different values selected for the intruder rulesets. The source is changed, otherwise the intruder would act as a passive network, in which case we do not actually need the adversary to forward the message. In this attack, an initiator pairs with an intruder using the public key of a responder. Since the intruder simply forwards messages with without changing the public keys, it cannot decode any messages sent between the initiator and responder once they're paired. However, since the initiator and responder are both paired with the intruder, the intruder can perform a denial-of-service attack by simply not forwarding the messages. This will clearly violate the initiator and responder authenticity invariant, since both think they are paired with the other, but are actually paired with an intruder.

```
-- intruder i sends recorded message
-- without modification
ruleset i: IntruderId do
  choose j: int[i].messages do
    ruleset k: AgentId do
      rule 100 ‘‘intruder sends recorded message’’
        -- Pick a message stored by an intruder
        !ismember (k, IntruderId) &
        !(int[i].messages[j].source = k) &
        multisetcount (l:net, true) < NetworkSize
      ==>
      -- Set destination to some AgentId,
      -- source to us, and send it out
      -- without further modifications
      var
        outM: Message;
      begin
        undefine outM;
```

```
        outM        := int[i].messages[j];
        outM.source := i;
        outM.dest   := k;
        multisetadd (outM, net);
      end;
    end;
  end;
end;
```

5.5.4 Modify and Forward

In this adversary model, the intruder can send any message over the communication channel to the other devices. When the intruder receives a message from an initiator, the intruder modifies the public key to its own public key before forwarding it to the intended responder. Similarly, when the intended responder sends its public key to the intruder, the intruder replaces it with the intruder's own public key before forwarding it to the initiator. In this way, both initiator and responder will be paired with the intruder, but unlike the "intercept and forward" model above, the intruder's public key is used. This means that the intruder can decode all messages between the initiator and responder, and contains a link key with both the initiator and responder. This model causes both the authentication and secrecy invariants for both initiator and responder to fail since not only are initiator and responder paired with the intruder, but link keys are shared with the intruder. Note that this MitM attack only occurs in the Just Works association models since in Numeric Comparison and Pass Key, users need to confirm the verification code before the protocol can continue.

```
-- intruder i modifies messages
-- and sends to network
ruleset i: IntruderId do
  choose j: int[i].messages do
    ruleset k: AgentId do
      rule 100  ''intruder sends recorded
                  message with its own info''
        -- Pick a message stored by an intruder
        !ismember (k, IntruderId) &
        !(int[i].messages[j].source = k) &
        multisetcount (l:net, true) < NetworkSize
      ==>
      var
        outM: Message;
      begin
        undefine outM;
        -- Modify the message depending on msg type:
        -- M_PublicKey, M_CommitValue,
        -- M_Nonce, or M_ExchangeVerif
        -- Send recorded message with modifications
        multisetadd (outM, net);
      end;
    end;
  end;
end;
```

5.6 Security Properties and Invariants

In order to verify that the Bluetooth 4.0 Pairing Protocol is secure we must translate some security properties to invariants for the Murphi model which check that the invariants are true at every state being explored by the model checker. The following subsections discuss the security properties as well as the invariants we implement.

5.6.1 Authentication

Bluetooth 4.0 provides authentication by providing commitment codes, verification codes to users, and finally an exchange code (all three done in phases 2 and 3 or the authentication steps in SSP). These steps are used to authenticate the two devices being paired. We define two authenticity invariants in our model such that no initiator is paired with an intruder and no responder is paired with an intruder if either of them are in the final paired states. The invariants are as follows:

```
-- initiator authenticity
invariant ''initiator correctly paired with
            good responder''
  forall i: InitiatorId do
    ini[i].state = I_PAIRED
    ->
    !ismember(ini[i].responder, IntruderId)
  end;

-- responder authenticity
invariant ''responders correctly paired with
            good initiator''
  forall i: ResponderId do
    multisetcount(l:res[i].pairings,
        (res[i].pairings[l].state = R_PAIRED &
         ismember(res[i].pairings[l].initiator,
                  IntruderId))
    ) = 0
  end;
```

5.6.2 Confidentiality

Confidentiality in Bluetooth 4.0 is provided through the encryption of the communication channel once the link key is established. This link key is refreshed periodically to ensure the confidentiality of communication. However, confidentiality is only upheld once the link key is established, all other communication before this is in the clear. As a consequence any eavesdropper can access the information being passed between the pairing devices; however, the information being passed is usually used for verification processes and therefore an adversary should not be able to gain all the information required to generate the paired link key. We thus represent this in our model by verifying that the adversary does not gain the link key of any user at any point in the analysis. Once the pairing is complete, if the adversary has the link key then the system has been compromised and thus we do not have to worry about the state of confidentiality after the generation of the link key as we assume all cryptographic primitives such as encryption used in the communication are secure. The invariants in our model thus check that once the initiators and responders are in the paired states that no intruder owns the honest agents' link keys.

```
-- initiator confidentiality
invariant ''initiator link key is secret''
  forall i: InitiatorId do
    ini[i].state = I_PAIRED &
```

```
    ini[i].linkKey = true
    ->
    forall j: IntruderId do
      int[j].linkKeys[i] = false
    end
  end;

-- responder confidentiality
invariant ''responder link key is secret''
  forall j: IntruderId do
    forall i: ResponderId do
      multisetcount(l:res[i].pairings,
        (res[i].pairings[l].initiator = j &
         res[i].pairings[l].state = R_PAIRED &
         res[i].pairings[l].linkKey = true &
         int[j].linkKeys[i] = true)
      ) = 0
    end
  end;
```

5.6.3 *Intention Preservation*

Intention preservation is very similar to authenticity; however, the major difference is the intention of the pairing and the invariant must hold such that whichever devices are originally in the pairing end up being paired. For example if an initiator intends to pair with an intruder's device then intention preservation states that at the end of the pairing the initiator is paired with the intruder's device whereas authenticy would not hold in this case. In order to verify this property we implement two invariants which check from the initiator's side and the responder's side of the protocol.

```
-- initiator intention preservation
-- every initiator device is paired with who
-- the device originally thought it
-- would be paired with
invariant ''initiator intention
            preservation pairing''
  forall i: InitiatorId do
    ini[i].state = I_PAIRED &
    ini[i].linkKey = true
    ->
    multisetcount(l:gpr,
        (gpr[l].initiator = i &
         gpr[l].responder = ini[i].responder)
    ) >= 1
  end;
```

```
-- responder intention preservation
-- every responder device is paired with who
-- the device originally thought it
-- would be paired with
invariant ''responder intention
            preservation pairing''
  forall i: ResponderId do
    multisetcount(j:res[i].pairings,
        res[i].pairings[j].state = R_PAIRED
    ) >= 1
    ->
    multisetcount(l:gpr,
      (multisetcount(j:res[i].pairings,
        res[i].pairings[j].state = R_PAIRED &
        gpr[l].responder = i &
        res[i].pairings[j].initiator
          = gpr[l].initiator
      ) >= 1)
    ) >= 1
  end;
```

5.7 Protocol Initial State

Our model begins with all states being cleared and the initiator and the responder both in the I_SLEEP or R_SLEEP states. All network and pairings are cleared, and the intruder only knows the information for itself and nothing else. Transition rules will then be fired off by the Murphi model checker that push these states to subsequent states and the validity checks will only continue along plausible paths of execution from this initial state. Since the intruder does not have any information in the beginning it must learn the information through our adversary model which was presented earlier.

6. ANALYSIS AND RESULTS

After running our Murphi model against our security invariants, our model discovered the following main attacks, two of which have been discussed in previous works.

6.1 Just Works Attack

In the Bluetooth 4.0 protocol using the JW association model our Murphi model verified an existing known MitM attack. In this attack the intruder intercepts a message from an initiator that was intended for an honest responder. The intruder then modifies the message with its own public key and sends this to the responder. The responder, not knowing that the intruder is initiating the pairing will respond back with its own public key in the exchange. The intruder then modifies this message and sends it to the initiator thus completing the public key exchange with both initiator and responder. From here the protocol continues as normal but since the JW association model does not have a verification check as in the NC model there is no detection of the MitM attack. Therefore even though the responder and the initiator have different Diffie-Hellman keys, the pairing will still occur. The intruder in the end has thus gained full control over the connection and can read everything that the initiator and responder send to each other.

Our formal modeling found this attack which broke the authenticity, confidentiality, and intention preservation invariants. This means that in the JW association model an intruder has a communication pairing with both devices, has indivudual link keys for the individual pairings, as well as broke the intention preservation by intercepting and forcing both devices to pair with itself.

6.2 Numeric Comparison Attack I

In the last section, we saw that the JW association model was suceptible to MitM attacks, which is one motivation for the NC association model in which the users of the devices would be presented with 6-digit verification codes to verify the connection. Thus when the intruder modifies the messages and exchanges its own public key PK_i with both devices the verification codes would be V_a =

$g(PK_a, PK_i, N_a, N_b)$ and $V_b = g(PK_i, PK_b, N_a, N_b)$. The users would only see the Device A which shows V_a and Device B which shows V_b which since we assume that g is a secure collision resistant hash function would produce two different values since the inputs are different. Thus the users would cancel the pairing and the MitM attack would fail. It is important to note that if there is user error involved and the users accept the verification codes despite being different then the MitM attack would succeed. Therefore the NC association model should protect against the MitM attack; however, as our model has found, it actually does not.

It is very important to understand that there aren't any proper authentication checks, the protocol relies on the built in exchanges to confirm a pairing but does not make any extent to actually verify the identity of the devices. This being said, our model discovered an MitM Impersonator attack in which the intruder impersonates an honest responder. To motivate this attack we discuss a real-world scenario. If an adversary takes a rental car and replaces the display with its adversarial device or modifies the car in such a way that the device appears to be the actual on-board device, then when an honest user enters the car they are tricked into believing that the device is an honest device. This scenario is similar to an ATM theif who places an external card reader over the actual card reader and unsuspecting users will input their ATM cards and have their card information swipped by the thief's card reader. This attack thus accomplishes an impersonation attack in which the user is tricked into forcing a pairing with the adversary. The pairing proceeds and now it is important to note that the user now sees Device A as well as the impersonated device and thus the verification codes would be $V_a = g(PK_a, PK_i, N_a, N_i)$ and $V_i = g(PK_i, PK_a, N_a, N_i)$ which would be the same and therefore the user will confirm the pairing and be paired with the intruder device. At this point the honest device responder does not need to be in the picture; however, if the intruder's device doesn't have access to certain things such as the on-board features, it could essentially create a connection with the honest device responder and impersonate being the honest user.

Our formal modeling verified the existence of this attack which breaks authenticity and confidentiality invariants, but not the intention preservation invariant. Thus the intruder pairs with the honest devices and knows the link key of the initiator and possible responder. The intention preservation invariant does not fail because the initiator intended to pair with the intruder device since the intruder was impersonating an honest responder.

6.3 Numeric Comparison Attack II

7. DISCUSSION AND FIXES

7.1 Just Works Attack

The JW association model is known to not provide protection against MitM attacks and is only present to allow for convenience for devices such as mice, keyboard, or headsets. For other devices, the NC association model is recommended for pairing such as for smartphones or laptops to help mitigate MitM attacks. Also for the general usages of Bluetooth, the range of the protocol is between 5-30 meters and therefore an adversary needs to be fairly close to the user in order to mount the attack, in which case it may not be as practical.

7.2 Numeric Comparison Attack

The NC association model is presented by the Core Bluetooth Specifications to protect against MitM attacks with high probability; however, our model verifies the previously discovered attack by Phan and Mingard [**?**] of the Impersonator MitM attack. This attack is very critical to understand because the NC association model is presumed to protect against MitM attacks; however, this doesn't take into consideration that the devices are not actually validated in anyway and therefore connecting to an impersonator device would result in a compromise of the system. Such an attack is difficult to fix against especially for a protocol that attempts at being energy efficient and lightweight; however, to flyly protect against such MitM attacks a trusted authority of some sort must be set up similar to the Certificate Authorities that are used on the Internet. Even then impersonation is difficult to defend against because there are many attack vectors that can exist such as also impersonating trusted entities in the verification propess, also the Bluetooth deves would then need to shave an eternal communication channel in oer to pto do proper verifications.

These attacks would be mostly successful in situations when a user must pair with a device that he or she does not own. In many scenarios, users often pair devices that they own because they trust them the most thus this attack although very critical is not necessarily a very critical threat model because of the disadvantages of having to trick a user into using an adversarial device and also being within vicinity of the user or at least get a device within their vicinity.

8. CONCLUSIONS

9. REFERENCES

[1] Bluetooth, SIG. Bluetooth core specification v4.0.30. `https://www.bluetooth.org/Technical/Specifications/adopted.htm`, June 2010.