Using Model-Checking to Reveal a Vulnerability of Tamper-Evident Pairing[⋆]

Rody Kersten[1], Bernard van Gastel[2], Manu Drijvers[1], Sjaak Smetsers[1], and Marko van Eekelen[1,2]

[1] Radboud University Nijmegen,
Institute for Computing and Information Sciences, The Netherlands
{r.kersten,s.smetsers,m.vaneekelen}@cs.ru.nl, manudrijvers@student.ru.nl
[2] Open University of the Netherlands, School of Computer Science, The Netherlands
{bernard.vangastel,marko.vaneekelen}@ou.nl

Abstract. Wi-Fi Protected Setup is an attempt to simplify configuration of security settings for Wi-Fi networks. It offers, among other methods, Push-Button Configuration (PBC) for devices with a limited user-interface. There are however some security issues in PBC. A solution to these issues was proposed in the form of Tamper-Evident Pairing (TEP). TEP is based on the Tamper-Evident Announcement (TEA), in which a device engaging in the key agreement not only sends a payload containing its Diffie-Hellmann public key, but also sends a hash of this payload in a special, trustedly secure manner. The idea is that thanks to the special way in which the hash is sent, the receiver can tell whether or not the hash was altered by an adversary and if necessary reject it.
Several parameters needed for implementation of TEP have been left unspecified by its authors. Verification of TEA using the Spin model-checker has revealed that the value of these parameters is critical for the security of the protocol. The implementation decision can break the resistance of TEP against man-in-the-middle attacks. We give appropriate values for these parameters and show how model-checking was applied to retrieve these values.

Keywords: Security, Model-checking, Spin, Wi-Fi Protected Setup, Tamper-Evident Pairing

1 Introduction

Security protocols aim at securing communications over networks that are publicly accessible. Depending on the application, they are supposed to ensure security properties such as authentication, integrity or confidentiality even when the network is accessible by malicious users, who may intercept and/or adapt existing, and send new messages. While the specification of such protocols is usually short and rather natural, designing a secure protocol is notoriously difficult.

[⋆] This work is part of the IOP GenCom GoGreen project, sponsored by the Dutch Ministry of Economic Affairs, Agriculture and Innovation.

Flaws are often found several years later. One of the sources for the vulnerability of such protocols is that their specification is often (deliberately) incomplete. There are several reasons for the omission of certain details by the designer. For instance, a protocol may depend on properties of the hardware on which it is used. It also leaves some room for the implementer of the protocol to make implementation-dependent choices. The problem with these unspecified parameters is that it can be very hard to analyze the effects of specific choices on the correctness of the protocol itself. Mostly this is due to the fact that the protocol is specified in such a way that both designer and implementer are either convinced that the correctness is not influenced by the concrete values of these parameters, or they assume that theses values are chosen within certain (not explicitly specified) boundaries.

During the last two decades, formal methods have demonstrated their usefulness when designing and analyzing security protocols. They indeed provide rigorous frameworks and techniques that allow to discover new flaws. For example, the ProVerif tool [4] and the AVISPA platform [1] are both dedicated tools for automatically analyzing security properties. More general purpose model-checkers, such as Spin [9] and Uppaal [3], are also successfully applied to verify desired properties of protocol specifications. While this model-checking process often reveals errors, the absence of errors does in general not imply correctness of the protocol.

Secure wireless communication is a challenging problem due to the inherently shared nature of the wireless medium. For wireless home networks, the so-called Wi-Fi Protected Setup was designed to provide a standard for easy establishment of a secure connection between a wireless device with a possibly limited interface (e.g. a webcam or a printer) and a wireless access point. The wireless device, once connected to the access point, gets not only internet connectivity, but also access to shared files and content on the network. The standard provides several options for configuring security settings (referred to as *pairing* or *imprinting*). The most prominent ones are PIN and Push-Button Configuration. The PIN method has been shown to be vulnerable to brute-force attacks; see [25]. This method and its weaknesses are briefly discussed in Section 5. To establish a secure connection using the Push-Button method, the user presses a button on each device within a certain time-frame, and the devices start broadcasting their Diffie-Hellman public keys [6], which are used to agree on the encryption key to protect future communication. In [8] the authors argue that this protocol only protects against passive adversaries. Since the key exchange messages are not authenticated, the protocol is vulnerable to an active man-in-the-middle (MITM) attack. To protect key establishment against these MITM attacks, [8] presents a method called *Tamper-Evident Pairing* (TEP), that provides simple and secure Wi-Fi pairing without requiring an out-of-band communication channel (a medium, differing from the communication channel that is used for transmitting normal data). The essence of their method is that the chip-sets used in Wi-Fi devices offer the possibility not only to transmit data, but also to sense the medium to detect whether or not information is communicated. The correctness of the proposed protocol is based on the assumption that an adversary can only change or corrupt data on the medium but not completely remove the data. The TEP protocol is specified in a semi-formal way; its correctness is proven manually (i.e. on paper; not formally using e.g. a theorem prover). However, in the protocol itself some parameters are used that are not fully specified.

In this paper we investigate the TEP protocol in order to determine whether its correctness depends on the values chosen for the unspecified parameters. In other words, we analyze the protocol by varying the values of these parameters in order to find out if there exists a combination for which correctness is no longer guaranteed. Our analysis is done by using the Spin model-checker. We have modeled the essential part of the protocol (known as the Temper-Evident Announcement), and used this model to hunt for potentially dangerous combinations of parameters, which indeed appeared to exist. The next step was to explore the vulnerability boundaries, by deriving a closed predicate relating the parameters to eachother and providing a safety criterion. The derivation of this predicate, and the verification of the resulting safety criterion, was done by using the model-checker. The contribution of our work is twofold. First, it reveals a vulnerability of a protocol that was 'proven to be correct'. And secondly, it shows how model-checking can be used, not only to track down bugs, but also to establish side-conditions that are essential for the protocol to work properly.

2 Tamper-Evident Pairing

The Wi-Fi Alliance has set the Wi-Fi Protected Setup (WPS) standard in [27]. The standard provides several options for simple configuration of security settings for Wi-Fi networks (pairing). One of them is Push-Button Configuration (PBC), where two devices (enrollee and registrar) are paired by pressing a (possibly virtual) button on each of the devices within a time-out period of two minutes. Security of this method is enclosed in the fact that the user needs physical access to both devices. However, in [8], three vulnerabilities are described creating opportunity for man-in-the-middle attacks:

1. **Collision:** An attacker can create a collision with the enrollee's message and send his own message immediately after.
2. **Capture effect:** An attacker can transmit a message at a much higher power than the enrollee. Capture effects were first described in [26].
3. **Timing control:** an attacker can occupy the medium, prohibiting the enrollee from sending his message, and send his own message in-between.

Gollakota et al also provide an innovative solution to the PBC security problems in [8]. Their alternative pairing protocol is named Tamper-Evident Pairing (TEP). It is based on the fact that Wi-Fi devices can not only receive packets, but also simply measure the energy on the channel, as part of the 802.11 standard requirements. This provides the opportunity to encode a bit of information as a time-slot where energy is present or absent on the wireless medium. Under the assumption that an attacker does not have the ability to remove energy from the medium, this means that an attacker cannot turn an on-slot into an off-slot.

Let us start by explicating the attacker model, i.e. the assumptions about the adversary that we are securing the protocol against. The presence of an active adversary is assumed, who is trying to launch a MITM attack. She has the following capabilities:

Overwrite data packets The adversary can use any of the three vulnerabilities listed above to overwrite data packets.

Introduce energy on the channel The adversary can introduce energy on the channel. Energy cannot be eliminated from the wireless medium.

2.1 The Tamper-Evident Announcement

To facilitate TEP, Gollakota et al introduce the Tamper-Evident Announcement (TEA) primitive, which is sent in both directions: enrollee to registrar and vice-versa. The structure of a TEA is given in Fig. 1. It starts with the so-called *synchronization packet*. This an exceptionally long packet, filled with random data. It is detected by the receiver by measuring a burst of energy on the medium of at least its length (so in a manner similar to the on-off slots). Because this packet is exceptionally long, this uniquely identifies a TEA.

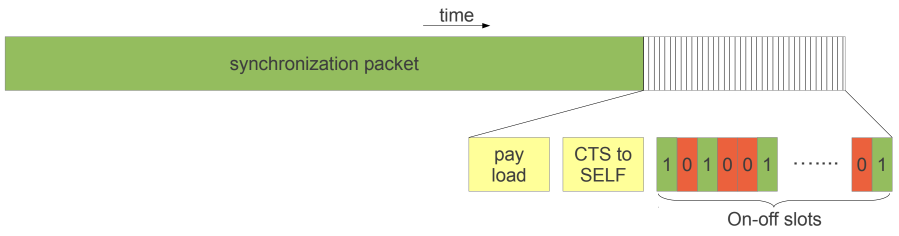

Fig. 1: The structure of a Tamper-Evident Announcement (TEA)

The synchronization packet is followed by the payload of the TEA, which contains the Diffie-Hellman public key [6] of the sender. Then, a *CTS-to-self* packet is sent. This message is part of the IEEE 802.11 specification and requests all other Wi-Fi devices not to transmit during a certain time period, here the time needed for the remainder of the TEA.

Finally, a hash of the payload is sent by either transmitting or refraining from transmitting during a series of so-called on-off *slots*. An attacker cannot change an on-slot into an off-slot, because she cannot remove energy from the medium, but she might still do the opposite. To be able to detect this as well, a specially crafted bit-balancing algorithm is applied to the 128-bit hash, prolonging it to 142 bits (71 zeros and 71 ones). Now, when an off-slot is changed into an on-slot, the balance between on and off slots is disturbed, making the tampering detectable. The 142-bit bit-balanced hash is preceded by two bits representing the direction of the TEA (enrollee to registrar or vice-versa). So, in total, 144 slots are sent.

2.2 Receiving the Slots

The sender sends out the 144 slots, which take 40 μs each, back-to-back. On the receiver-side the slots are received by measuring energy on the wireless medium. The receiver iteratively measures the energy on the medium, during so-called *sensing windows* of 20 μs. The total number of measurements m during the sensing window is stored, as well as the number of measurements e during which there was energy on the medium. If the *fractional occupancy*, given by e/m, is above a certain threshold then the medium is considered occupied during this particular sensing window.

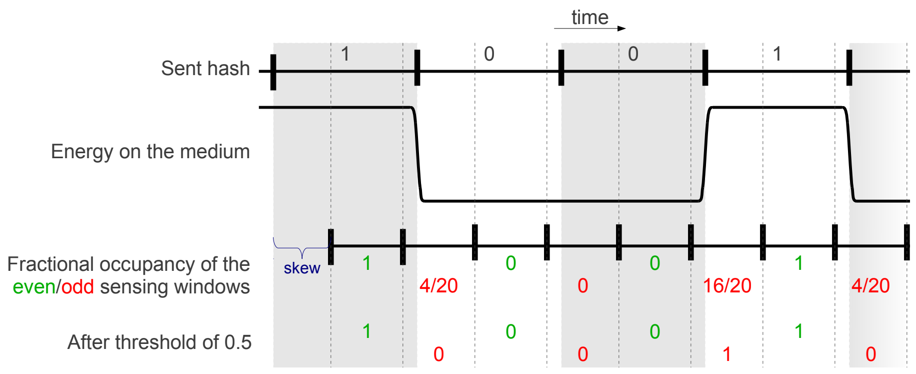

Fig. 2: Sending and receiving the slots of a 4-bit hash. The even sensing windows have the higher variance here. Therefore, those represent the received hash. Clock skew is shown in blue on the left.

The length of a sensing window is half the slot-length. The reason for this is that now either all the even sensing windows or all the odd sensing windows fall entirely within a slot, i.e. do not cross slot-boundaries, shown in Fig. 2, where the even sensing windows all fall entirely within one of the 40 μs slots. Note that the figure shows the ideal case, where measurements are exact. In reality the measurements will be less than perfect, which motivates the use of a threshold. The use of this special method of receiving the slots is motivated by the fact that there may be a slight clock-skew. This is shown in Fig. 2 on the lower-left.

After all the measurements are done and after applying the threshold, the receiver verifies that either the even or the odd sensing windows have an equal number of zeros and ones[3], and that those match a calculated hash of the payload packet. If this is not the case, then the receiver aborts the pairing process.

3 Modeling the Tamper-Evident Announcement in Spin

We use the same attacker model as the authors of [8], listed in Sect. 2. Given that an adversary can replace the payload packet, we will try to verify that she cannot adapt the bits of the hash that are received without being detected. Namely, if the attacker manages to send her own payload and adapt the hash such that it matches her payload and contains an equal number of zeros and ones, she can initiate a MITM attack. The payload packet itself is therefore not part of the model. We will only model the sending of the bit-balanced hash. The direction bits (i.e. the first two slots) are also not modeled. Gollakota et al. give an informal proof of the security of TEP in [8]. Effectively, we are challenging Proposition 7.2 of their proof.

We have used Spin [9] for the verification of the model. This section contains some illustrative simplified fragments from the model only. The full model (including results) can be downloaded from `http://www.cs.ru.nl/R.Kersten/publications/nfm/`.

3.1 Model Parameters

The model has a series of parameters that are described in this section.

Hash length The length of the bit-balanced hash to send. All possible hashes of this length that are bit-balanced are tried (the balancing algorithm itself is not part of the model).

Number of measurements per sensing window The number of measurements in each sensing window depends on the Wi-Fi hardware on which the protocol is implemented. The length of the window is 20 μs. During each window, the hardware logs the total number of clock-ticks, as well as the number of clock-ticks during which there was energy on the wireless medium. The number of measurements during each sensing window is thus variable. In the model though, the number of measurements is fixed and given by a parameter. The reason for this is that a variable number of measurements would highly enlarge the state-space, the number of measurements is not something that an adversary can influence and that we believe it will be fairly constant in practice. A programmer implementing the protocol could measure or calculate the average number of measurements during a sensing window and use a "safe" approximation (a little lower) in the formula. In

[3] Actually, the variance of all the even sensing window measurements and that of all the odd sensing window measurements is calculated. The sensing windows with the higher variance will be the correct ones, since on and off slots are balanced. It is however not clear to us what the advantage of this approach is over simply selecting the sensing windows in which the on-off slots are balanced.

our model, the sender puts energy on the wireless medium for the number of clock-ticks it takes to do the measurements for two sensing windows (the sensing window has half the length of an on-off slot). This means that one measurement is the unit for a clock-tick.

Sensing window threshold As explained in Sect. 2.2, bits are received by measuring the *fractional occupancy* during a sensing window. It is determined whether or not the medium was occupied in a sensing window by checking if the fractional occupancy is above a certain threshold. The value of this threshold is not defined in [8], although it influences the measurements heavily. Since the number of measurements during a sensing window is constant in the model, we can omit the calculation of the fractional occupancy. This means that also the threshold should now be modeled, not as a number between 0 and 1, but as a number between 0 and the number of sensing window measurements and that its unit is clock-ticks (the medium was occupied during e ticks of the discrete clock). If the number of measurements (clock-ticks) in a sensing window where there was energy on the medium exceeds the threshold, then a one is stored for this sensing window.

Skew The reason for the use of pairs of sensing windows for receiving the slots is that there may be an inherent clock skew. It is stated in [8] that this inherent clock skew may be up to 10 μs, i.e. half the sensing window length. By using pairs of sensing windows, either the even or the odd windows are guaranteed not to cross slot-boundaries. Furthermore, it is stated in [8] that to detect a TEA it is sufficient to detect a burst of energy "at least as long as the synchronization packet". It is not specified which is the exact synchronization point: the beginning or the end of the energy pulse. Neither is the maximum length of an energy burst that signifies a synchronization packet. The difference with the given length of 19ms introduces an extra skew. Since an adversary can introduce energy to the wireless medium, she can prolong the synchronization packet and introduce extra skew (the sign of this skew depends on the choice of synchronization point). The model variable *skew* is the total of the inherent clock skew and this *attacker skew*. Like the number of measurements and the threshold, its unit is also clock-ticks. We only consider positive skew (forward in time) in our model.

These parameters to the model are henceforth referred to as *hash_length*, *sw_measurements*, *threshold* and *skew*, respectively.

3.2 Clock Implementation

Timing is essential to modeling the TEA. However, Spin has no inherent notion of time. Luckily, in this case the exact scheduling and execution speed are not important, as the only interaction between the sender and receiver processes is sending energy to and reading the energy-level from the wireless medium. The receiver observes the value of the wireless medium once per clock cycle, the sender updates it at most once.

Due to these properties we can implement a discrete clock in Promela (the modeling language used by Spin), without the need to use specialized model-checkers with native clock support. We introduce a separate clock process, which waits until all processes using a clock are finished with a clock cycle (Listing 1, line 17), before signaling them to continue. Processes are signaled to continue by flipping the Boolean *clock* (line 23). Processes can only continue with the next clock cycle if this variable differs from their local variable *localclock* (line 10), which is also flipped after each clock-tick (line 11). Our clock implementation also supports processes which do not use a clock. A clock-tick in the model corresponds to a measurement taken by the receiver. To avoid the situation that the receiver executes before the sender, we implemented explicit turns for the processes, so the sender always executes first after a clock-tick. The process with the lowest process identifier is always executed first (line 7). We can introduce skew by letting one of the processes wait a number of clock-ticks before starting.

```
 1  byte waiting = 0;
 2  bool clock = false;
 3  #define useClock() bool localclock = false;
 4
 5  inline waitTicks(procID, numberOfTicks) {
 6    byte tick;
 7    for (tick : 0..(numberOfTicks-1)) {
 8      waiting++;
 9      atomic {
10        localclock != clock;
11        localclock = clock;
12        waiting==procID;
13      }
14    }
15  }
16
17  proctype clockProc() {
18  end:
19    do
20    :: atomic {
21      waiting==NUMBER_OF_CLOCK_PROCESSES;
22      waiting = 0;
23      clock = !clock;
24    }
25    od;
26  }
```

Listing 1: Modeling the clock. The `useClock` and `waitTicks` functions must be used in processes that use the clock.

3.3 Model Processes

The model begins with a routine that generates all possible hashes of the given length non-deterministically. It then starts four processes:

Clock A simple clock process is used to control the other processes. The clock process is described in Sect. 3.2.

Sender The sender process first initializes and starts the clock. It then iteratively sends a bit of the generated hash (by putting energy on the medium, or not), waits for $2 \cdot sw\_measurements$ clock-ticks, then sends the next bit. When finished sending, the sender must keep the clock ticking, because the receiver process might still be running.

```
proctype sender() {
  useClock();
  waitTicks(0, 1);
  byte i;
  for (i : 0..(HASH_LENGTH-1)) {
    mediumSender = get(i); //send slot
    waitTicks(0, SW_MEASUREMENTS*2);
  }
  doneWithClock(0);
}
```

Listing 2: Sender model.

Receiver The receiver also begins with initializing and starting the clock. It then introduces clock skew by waiting more *skew* ticks. Then, it measures energy on the medium *sw_measurements* times (once each clock-tick) and stores the received bit for each sensing window (one if e is above *threshold*). Note that the wireless medium consists of two bits: one that is set by the legitimate sender and one that is set by the adversary. The receiver reads energy if either bit is set. Once measurements for all sensing windows are done, the `checkHash()` function verifies if either the even or the odd sensing windows have an equal number of on and off slots.

```
receiver() {
  useClock();
  waitTicks(1, SKEW+1);
  short sw;
  for (sw : 0..(HASH_LENGTH*2-1)) {
    byte e = 0, ticks = 0;
    for (ticks : 0..(SW_MEASUREMENTS-1)) {
      e = e + (mediumSender || mediumAdversary);
      waitTicks(1, 1);
    }
    store(sw, e>THRESHOLD);
  }
  checkHash();
}
```

Listing 3: Receiver model.

Adversary The adversary is modeled as a simple process that increases the energy on the medium, then decreases it again. Because processes may be interleaved in any possible way, this verifies all scenarios.

```
proctype adversary() {
end:
  do
  :: mediumAdversary = 1;
     mediumAdversary = 0;
  od
}
```

Listing 4: Adversary model.

4 Model-Checking Results

Verification of the model means stating the assertion that either the received hash is equal to the sent hash or it is not equal, but the tampering by the adversary is detected (because the number of ones in the hash is unequal to the number of zeros). It is thus a search for a counter-example.

The expectation was that we might be able to find such a counter-example, but that the freedom with which an adversary could modify the received hash would be limited, probably to just the first or last bit. Model-checking indeed generated a counter-example. Moreover, experimentation with different assertions turned out that the adversary actually has more freedom in modifying the hash than expected. This vulnerability is described in Sect. 4.1. After the vulnerability was discovered, we executed a large series of Spin runs to discover what the exact conditions are that enable such an attack. The results are given in Sect. 4.2.

4.1 Revealed Vulnerability in the TEA

Model-checking the TEA model using Spin generated a counter-example to the assertion that a hash that was modified by an adversary will not be accepted by the receiver. A scenario similar to the one for which this counter-example was found is shown in Fig. 3.

Figure 3 shows the case where no adversary is active. Here, the even sensing windows still have the higher variance (1001 versus 0010). Thus, those are chosen as the correct slots and the hash 1001 is received, which is equal to the sent hash.

In Fig. 4 a scenario is shown in which an adversary actively introduces energy on the wireless medium. The energy that is introduced by the attacker is shown as a dotted blue line. She manages to trick the receiver into choosing the odd sensing windows and consequently receive a modified hash: 1010. Experimentation with modified assertions has confirmed our conjecture that an adversary can use this tactic to change any 1 bit in the hash to a 0, *if and only if it is immediately followed by a 0.* Since the hash is bit-balanced, it consists of 50% zeros and 50% ones. Of the latter category, half are followed by 0 bits on average. This means that an adversary can change on average 75% of the hash.

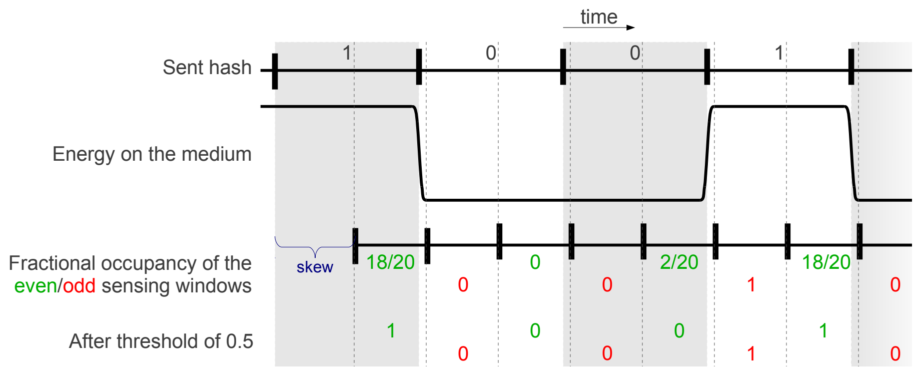

Fig. 3: Scenario in which TEP is vulnerable, for a 4-bit hash. The synchronization packet is prolonged to create a skew that is larger than the half the sensing window. The hash is still received correctly here.

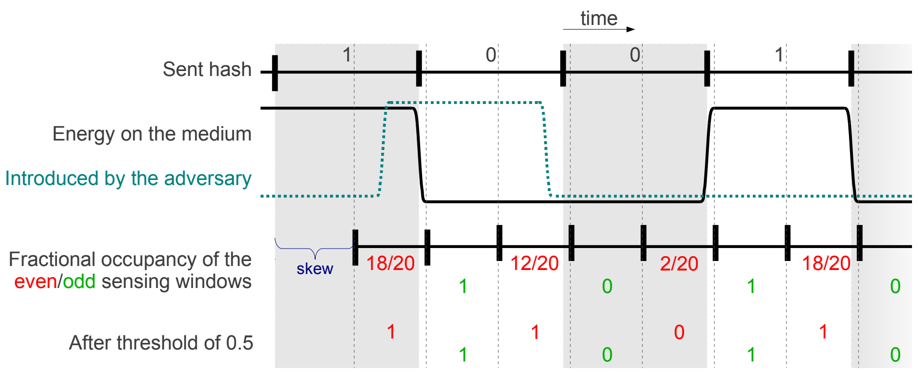

Fig. 4: The attack found by model checking, for a 4-bit hash. The adversary introduces energy to the medium to change the received hash to 1010.

4.2 Varying the Values of the Model Parameters

After discovering the vulnerability described in the previous section, we wanted to investigate what the exact circumstances are in which the vulnerability occurs. We therefore ran the Spin model-checker for many different values of the parameters *hash_length*, *sw_measurements*, *threshold* and *skew*[4]. Some of the results are shown in Table 1. As it turns out, the length of the hash has no influence on the occurrence of the vulnerability, so this is omitted from the results. Remember that the unit for all three parameters in the table is clock-ticks.

[4] In order to run the Spin model-checker for various values of defined parameters, we have implemented a small wrapper in the form a of C program. This wrapper can be obtained from `http://www.open.ou.nl/bvg/spinbatch/`.

$threshold = 3$		*skew*										
		0	1	2	3	4	5	6	7	8	9	10
sw_meas.	4	+	-	-	-	-	-	-	-	-	-	-
	5	+	+	-	-	-	-	-	-	-	-	-
	6	+	+	+	-	-	-	-	-	-	-	-
	7	+	+	+	+	-	-	-	-	-	-	-
	8	+	+	+	+	+	-	-	-	-	-	-
	9	+	+	+	+	+	+	-	-	-	-	-
	10	+	+	+	+	+	+	+	-	-	-	-

(a) Results for *threshold* = 3.

$threshold = 5$		*skew*										
		0	1	2	3	4	5	6	7	8	9	10
sw_meas.	6	+	-	-	-	-	-	-	-	-	-	-
	7	+	+	-	-	-	-	-	-	-	-	-
	8	+	+	+	-	-	-	-	-	-	-	-
	9	+	+	+	+	-	-	-	-	-	-	-
	10	+	+	+	+	+	-	-	-	-	-	-

(b) Results for *threshold* = 5.

$threshold = 7$		*skew*										
		0	1	2	3	4	5	6	7	8	9	10
sw_meas.	8	+	-	-	-	-	-	-	-	-	-	-
	9	+	+	-	-	-	-	-	-	-	-	-
	10	+	+	+	-	-	-	-	-	-	-	-

(c) Results for *threshold* = 7.

$threshold = 9$		*skew*										
		0	1	2	3	4	5	6	7	8	9	10
sw_meas.	10	+	-	-	-	-	-	-	-	-	-	-

(d) Results for *threshold* = 9.

Table 1: Model-checking results. Pluses indicate that the proposition is not broken. Minuses indicate the occurrence of the vulnerability.

It is obvious from Table 1 that the following predicate determines the possibility of an attack:

$$skew \geq sw\_measurements - threshold \tag{1}$$

In Fig. 4, a threshold of 0.5 is used, which is represented by a value for *threshold* of $\frac{1}{2} \cdot sw\_measurements$ in the model. If the *skew* is large enough to move a number of *threshold* measurements of the even windows over the sensing window boundary, then an adversary might change the received hash. We have model-checked the predicate for all combinations of *sw_measurements* 1–10, *threshold* 1–10 and *skew* 1–10.

5 Related Work

Before the Spin model on which this article is based was made, a simple model of the TEA and TEP in UPPAAL was made by Drijvers [7]. UPPAAL is a tool with which properties about systems modeled as networks of timed automata can be verified [3]. Because of the simple nature of this model, it did not include clock skew and therefore did not reveal the vulnerability that was later found using Spin. Apart from the TEA, Drijvers made a separate model of the overlying TEP, with which – under the assumption that the TEA is secure – no problems were identified. Since TEP was already successfully model-checked using UPPAAL and, contrary to the TEA model, not in a highly abstract form (it is much simpler), we chose not to repeat the modeling for Spin.

In [5] a method is proposed for modeling a discrete clock in Promela, without the need to alter Spin. Instead of an alternating Boolean, time is modeled as an integer which negatively impacts the state space explosion. Just as with our approach a separate clock is introduced, which waits until all the other processes are finished, before increasing the discrete clock variable. This waiting is modeled with a native feature of Promela, which only continues if no other state-transition can be made (the `timeout` keyword). Therefore, all the processes are implicitly using the modeled clock. Because of our general adversary process, this restriction is too severe for us.

Many approaches to pairing wireless devices are described in the literature. A comparison of various wireless pairing protocols is given in [24]. Often, a trusted *out-of-band* channel is used to transfer (the hash of) an encryption key, e.g. a human [10], direct electrical contact [23], Near-Field Communication [17], (ultra)sound [16], laser [18], visual/barcodes [20], et cetera. A nice overview is given in [11]. Another, slightly out-dated, overview is given in [22]. In TEP, a hash of the key is communicated in a trustedly secure manner *in-band.*

When using the PIN method that Wi-Fi Protected Setup provides, one of the devices displays an eight-digit authentication code, which the user then needs to enter on the other device. This method thus requires a screen and some sort of input device. The PIN method has been shown to be vulnerable to feasible brute-force attacks by Viehböck in [25]. The reason for this is that last digit is actually a check-sum of the first seven digits (i.e. there are only seven digits to verify) and, moreover, that the PIN is verified in two steps. The result of the verification of the first four digits is sent back to the enrollee, which may then send three more digits if this result was positive. This reduces the number of codes to try in a brute-force attack from 10^7 to $10^4 + 10^3$. A successful attack can be executed in approximately two hours on average. CERT-CC has urged users to disable the WPS feature on their wireless access points in response to this vulnerability[5]. A security and usability analysis of Wi-Fi Protected Setup, as well as Bluetooth Simple Pairing, which is similar, is given in [12].

Approaches to model-checking security protocols are described in [13] and [2]. In [14], a series of XSS and SQL injection attacks is detected using model-checking. Model-checking and theorem proving of security properties are discussed in [15]. In [19], security issues that arise from combining hosts in a network are investigated using model-checking. An entire Linux distribution is model-checked against security violations in [21].

6 Conclusions

The effects of a number of decisions to be made when implementing Tamper-Evident Pairing have been studied. In particular, the sending of a hash by using on-off slots – in which energy is present or absent on the wireless medium – was modeled. The values of several essential parameters of the protocol have not

[5] `http://www.kb.cert.org/vuls/id/723755`

been adequately specified. Model checking proved to be very effective both in uncovering a serious vulnerability for certain values of these parameters and in finding a predicate on the parameters indicating for which values the vulnerability is present. An adversary aiming to initiate a man-in-the-middle attack can thus *evidently tamper* with the received hash.

Future work could include extending the model to cover more of the TEA and investigate the feasibility of exploiting the found vulnerability. Furthermore, a full formal proof that the found vulnerability cannot occur when the predicate is not satisfied would be very valuable.

References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Drielsma, P. Hem, O. Kouchnarenko, J. Mantovani, S. Mdersheim, D. Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigan, and L. Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer Berlin Heidelberg, 2005.
2. A. Armando, R. Carbone, and L. Compagna. LTL model checking for security protocols. *Journal of Applied Non-Classical Logics*, 19(4):403–429, 2009.
3. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, Oct. 1995.
4. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.
5. D. Bošnacki and D. Dams. Integrating real time into Spin: a prototype implementation. In D. Bošnacki, editor, *Enhancing State Space Reduction Techniques for Model Checking*. Technische Universiteit Eindhoven, 1998.
6. W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644 – 654, nov 1976.
7. M. Drijvers. Model checking Tamper-Evident Pairing. Bachelor thesis, Radboud University Nijmegen, 2012.
8. S. Gollakota, N. Ahmed, N. Zeldovich, and D. Katabi. Secure in-band wireless pairing. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association.
9. G. Holzmann. The model checker Spin. *Software Engineering, IEEE Transactions on*, 23(5):279 –295, may 1997.
10. R. Kainda, I. Flechais, and A. W. Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 11:1–11:12, New York, NY, USA, 2009. ACM.
11. A. Kobsa, R. Sonawalla, G. Tsudik, E. Uzun, and Y. Wang. Serial hook-ups: a comparative usability study of secure device pairing methods. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 10:1–10:12, New York, NY, USA, 2009. ACM.

12. C. Kuo, J. Walker, and A. Perrig. Low-cost manufacturing, usability, and security: An analysis of Bluetooth Simple Pairing and Wi-Fi Protected Setup. In S. Dietrich and R. Dhamija, editors, *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2007.
13. G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 96 –105, jun 1998.
14. M. Martin and M. S. Lam. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 31–43. USENIX Association, 2008.
15. F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 44 –52, 1998.
16. R. Mayrhofer and H. Gellersen. On the security of ultrasound as out-of-band channel. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –6, march 2007.
17. R. Mayrhofer, H. Gellersen, and M. Hazas. Security by spatial reference: Using relative positioning to authenticate devices for spontaneous interaction. In J. Krumm, G. Abowd, A. Seneviratne, and T. Strang, editors, *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 199–216. Springer Berlin Heidelberg, 2007.
18. R. Mayrhofer and M. Welch. A human-verifiable authentication protocol using visible laser light. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 1143 –1148, april 2007.
19. R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156 –165, 2000.
20. N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp. –313, may 2006.
21. B. Schwarz, H. Chen, D. Wagner, G. Morrison, J. West, J. Lin, and W. Tu. Model checking an entire Linux distribution for security violations. In *Computer Security Applications Conference, 21st Annual*, pages 10–22, 2005.
22. D. B. Smetters, D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless ne tworks. 2002.
23. F. Stajano and R. Anderson. The resurrecting duckling: security issues for ubiquitous computing. *Computer*, 35(4):22 –26, apr 2002.
24. J. Suomalainen, J. Valkonen, and N. Asokan. Security associations in personal networks: A comparative analysis. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *Security and Privacy in Ad-hoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 43–57. Springer, 2007.
25. S. Viehböck. Brute forcing Wi-Fi protected setup. http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf.
26. C. Ware, J. Judge, J. Chicharo, and E. Dutkiewicz. Unfairness and capture behaviour in 802.11 adhoc networks. In *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, volume 1, pages 159 –163 vol.1, 2000.
27. Wi-Fi Alliance. *Wi-Fi Protected Setup Specification, version 1.0h*, 2006.