Some Security Considerations over Contiki-based Sensor Network

Yan Yan

December 18, 2015

Contents

Chapter 1

Introduction

This paper discusses two security measurements, namely Link Layer Security (LLSEC) and Datagram TLS (DTLS), within Contiki OS.

In Chapter 2 we described a LLSEC implementation in Contiki OS *noncoresec* and argues that its selection of IV is not secure enough and potentially has a reset problem.

Chapter 3 discusses some implementation issues of DTLS on Contiki OS.

Chapter 4 first argues that under some natures of WSN, an adversary could possibly collect more accurate timing information than usual Internet Web-application attacker. Then we described a potential sensor node application fingerprinting attack using the interaction of PING protocol and the application running on the target node.

1.1 Related Work

[1] discusses some security concerns in 802.15.4. LLSEC[2] is the implementation of 802.15.4 security in Contiki.

tinydtls[3] is the implementation of DTLS we used in DTLS related experiments.

1.2 Experiment Setup

All experiments are done within the Cooja simulator.

The setup is as described in Figure 1.1.

- **Adversary** is the malicious party that tries to recover information from the encrypted traffic.

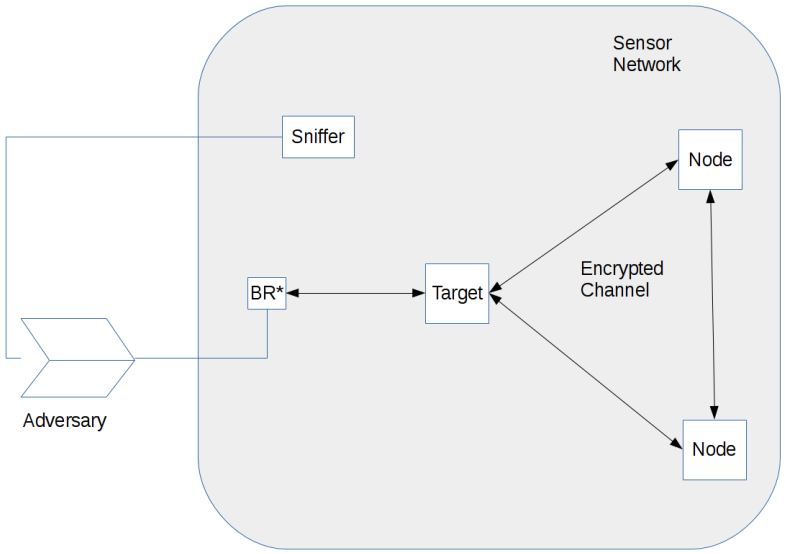

Figure 1.1: Experiment setup

- **Border Router**, or BR, is a device that connects the adversary to the sensor network. However, **BR is not allowed when LLSEC is enabled** as the adversary does not have the key and hence cannot connect into the network.
- **Sniffer** is a device that passively captures all traffics in the sensor network.
- **Target** and **Nodes** are sensors deployed in the sensor network. They communicates to each other through encrypted channels.
- **Sensor Network** discussed in this paper is a 6LowPAN network.

1.3 Adversary Power

The powers assumed in the experiments are considered to be practical in real life.

When LLSEC is enabled, all traffic, including RPL[1] messages, are encrypted; therefore no external nodes can connect to the network. The only power for the adversary is to passively sniff all the traffic.

In other cases where LLSEC is disabled, the adversary will be enabled to join the sensor network through a BR and hence is also capable to send ICMP messages to the target(s).

[1]Routing Procol for Low-power and Lossy Networks

1.4 Types of Packets

We simply categorise the packets into two types:

- **Network Management Packets**: These are the packets generated by the protocols those maintains the network, such as MAC ACKs, RPL messages or ICMP messages.
- **Data Packets**: These are those packets generated by the applications running on the nodes., such as a CoAP packet.

This is only a subjective rough categorisation and may not be precise. For example an TCP data packet may also serves as an ACK, or DTLS handshake packets could fall into both categories. However, we ignore this ambiguity as it is not our focus.

Chapter 2

Link Layer Security

Link Layer Security, or LLSEC, is a security measure that implements cryptography just above the physical layer.

Introducing cryptography at a lower level has several benefits. Firstly, more data being encrypted reduces the observable packet features to an adversary, such as SRC[1] and DST[2] field in the IP header which are very likely to be exploited by the adversary. Secondly, authentication at lower level also prevents an active adversary from joining the network which therefore weakens his power.

Imposing cryptography at a lower level also brings more challenge to the design of sensor network architecture. The first problem is its overhead. Even for a node that only tries to retransmits the packet to its next hop, it must decrypt the whole packet to extract its routing information, and then re-encrypt it before retransmission. This is particularly problematic in a mesh wireless sensor network as it could potentially lead to performance and energy consumption problems. Key management is also challenging due to the lossy and power optimised nature of wireless sensor network.

It is also noticeable that some packet features are not hidden even with LLSEC enabled, such as packet length, timing information and part of the MAC header.

2.1 802.15.4 Security: *noncoresec*

noncoresec[2] is the current implementation of LLSEC in Contiki. It corresponds to the AES_CCM_16 ciphersuite in 802.15.4 standard. This section briefly describes how it works.

- **Key Management**: All nodes share a network wide AES key for both encryption and authentication. The key is hardcoded during the setup stage.

[1]Source Address
[2]Destination Address

Flags(1)	Addresses(8)	Frame Counter(4)	Security Level(1)	Block Counter(2)

Table 2.1: IV of 802.15.4 Frame with Security

- **AEAD**[3]: *noncoresec* implements AES_CCM_16 [4] as described in 802.15.4[4]. CCM mode turns AES into a stream cipher. The same key is used for both encryption and authentication.
- **Initial Vector (IV, or nonce)**: The IV for each packet is constructed from certain fields of unencrypted MAC frame header and therefore is public.

An adversary without the knowledge cannot join the sensor network as he cannot sent out a valid RPL message.

2.2 Weak IV

One problem within the *noncoresec* implementation is the low variance of IV. The IV is a 16 byte bit-string constitutes of the following fields(Table 2.1):

- **Flags (1 byte)**: This field contains part of the MAC frame header. It is identical to most (basically all) of the data packets.
- **Source Address (8 bytes)**: This is mapped from the source address field of the frame.
- **Frame Counter (4 bytes)**: This field increases by 1 for each frame sent.
- **Security Level (1 byte)**: This field indicates which ciphersuite to be used for this frame. In the case of AES_CCM_16, this is constantly 0x7.
- **Block Counter (2 bytes)**: This field begins from 0x0 and increases by 0x1 for each block in CCM mode. The block length for AES-128 is 16 bytes. The 2 bytes counter is sufficient as it supports up to 2^{32} bytes of data whereas the minimum MTU[5] required by 6lowPAN standard[5] is 127 bytes.

In the current *noncresec* implementation, **Flags** and **Security Level** are constant. **Block Counter** always begins from 0x0 and the **Source Address** is also constant for a specific device. Such design leaves the 4 bytes **Frame Counter** the only field that is variable. This indicates that only 2^32 messages are allowed which is cryptographically considered to be inappropriate.

[3]Authenticated Encryption with Associated Data
[4]CCM mode of AES-128 with 16 bytes MAC
[5]Maximum Transmit Unit, simply speaking this is the maximum length of a packet.

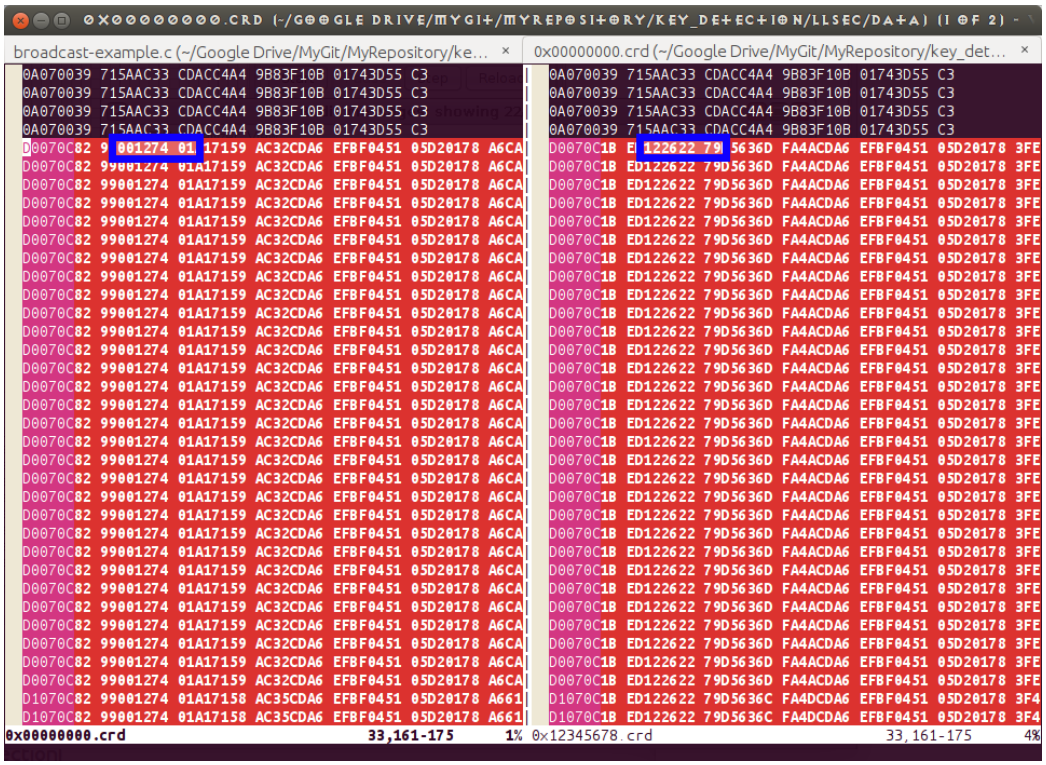

Figure 2.1: Captured packets with *noncoresec* enabled

2.2.1 Reset Problem

The low variance of IV leads to a plaintext leakage problem which only requires the adversary to reboot the target node.

The idea is that rebooting the device resets the **Frame Counter** to 0x0; hence once a pair of packets with same **Frame Counter** is found, the difference of their plaintext can be computed by their ciphertext:

$$\Delta p = c_1 \oplus c_2$$

where Δp is the difference of plaintexts. c_1 and c_2 are their ciphertext respectively.

Example 2.2.1. Figure 2.1 demonstrates some packet captured[6] with *noncoresec* enabled. These packets are captured with a sensor broadcasting a 4 byte integer with left side of Figure 2.1 being $[00000000]_{16}$ and right $[12345678]_{16}$. Marked are the corresponding ciphertexts which are $[00127401]_{16}$ and $[12262279]_{16}$ respectively.

[6]The duplicated packets are caused by the retransmission of ContikiMAC[6].

As we can see, the difference of ciphertext is exactly the difference of plaintext:

$$\Delta p = [00127401]_{16} \oplus [12262269]_{16} = [12345678]_{16} \tag{2.1}$$

2.3 Distinctive packet length for RPL packets

Some RPL packets are shorter than the minimum length of data packets which can be used to distinguish the packets.

2.4 Performance issue

The header overhead with LLSEC enabled is 20 bytes which is relatively a large overhead comparing to the 127 bytes MTU requirement of 6LowPAN standard[5].

Chapter 3

DTLS

DTLS has potentially the best interoperability as it is an variation of the widely used TLS in Internet. However, its design might not fit into the nature of WSN for practical reasons.

3.1 Implementation Issues

tinydtls[3] currently supports two ciphersuites, namely TLS_PSK_WITH_AES_128_CCM_8 and TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8.

However, we encountered several difficulties when trying to set up a network with DTLS.

Low Computational Power
Curve computation requires relatively a large amount of computational power. Even using a relatively power platform (CC2538), it still takes minutes to complete a DTLS handshake with TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8.

Low Bandwidth
The 6LowPAN standard specifies that the minimum MTU is 127 bytes whilst 67 (87 with LLSEC) bytes are occupied by protocol headers until UDP, which leaves 60 (40 with LLSEC) bytes available for UDP layer payload. This value can be easily exceeded even during the handshake, such like using a longer key. Some attempts have been made to solve this issue, e.g. CoDTLS[7][1].

Code Size
The tinydtls fails to fit into some devices, e.g. skymote, as its size of code is too large.

[1]This draft has been abandoned for some reason we do not know.

Therefore although TLS_PSK_WITH_AES_128_CCM_8 is less flexible (and probably less secure) as it uses a pre-shared master secret than TLS_ECDHE_ECDSA_WITH_AES_128_CCM it is still considered to be a relatively practical security measure as it requires less resources.

3.2 No multicast support

Some application protocols, such as CoAP, utilises the multicast feature of 6LowPAN whilst TLS is a protocol designated for securing communications between two parties, so is DTLS.

3.3 Overloading DTLS with LLSEC

Adopting both security measures at the same time is possible as they are implemented at different layers. However, it is questionable whether this will bring more security, as both *noncoresec* and TLS_PSK_WITH_AES_128_CCM_8 are using 128 bit AES with CCM mode as their cryptographic primitive.

Chapter 4

Application Detection

Similar to website fingerprinting, we try to identify the application running on target note by its traffic. This chapter discusses some general idea without a specific application.

4.1 Network Protocol Headers

Since most information in MAC[1], IP and UDP headers are related to routing and network maintenance and thus independent except the length fields and CRC[2].

4.2 Packet Length

Packet length is usually the most interested target in traffic analysis. However, packet lengths are also highly application dependant; thus we are not pursuing this topic further without a specific application.

4.3 Timing Packet Response

Unlike web applications where the client and server are usually physically distant, sensor networks can sometimes located in a concentrated area, such as a house which its radius can be less than 10 meters.

These features theoretically enables one to capture all traffics in such a sensor network. As opposed to the case of Internet where packets are usually timed on

[1]Media Access Control, not to be confused with the cryptographic term Message Authenticate Code.

[2]Cyclic Redundancy Check, a code to detect or correct transmission error

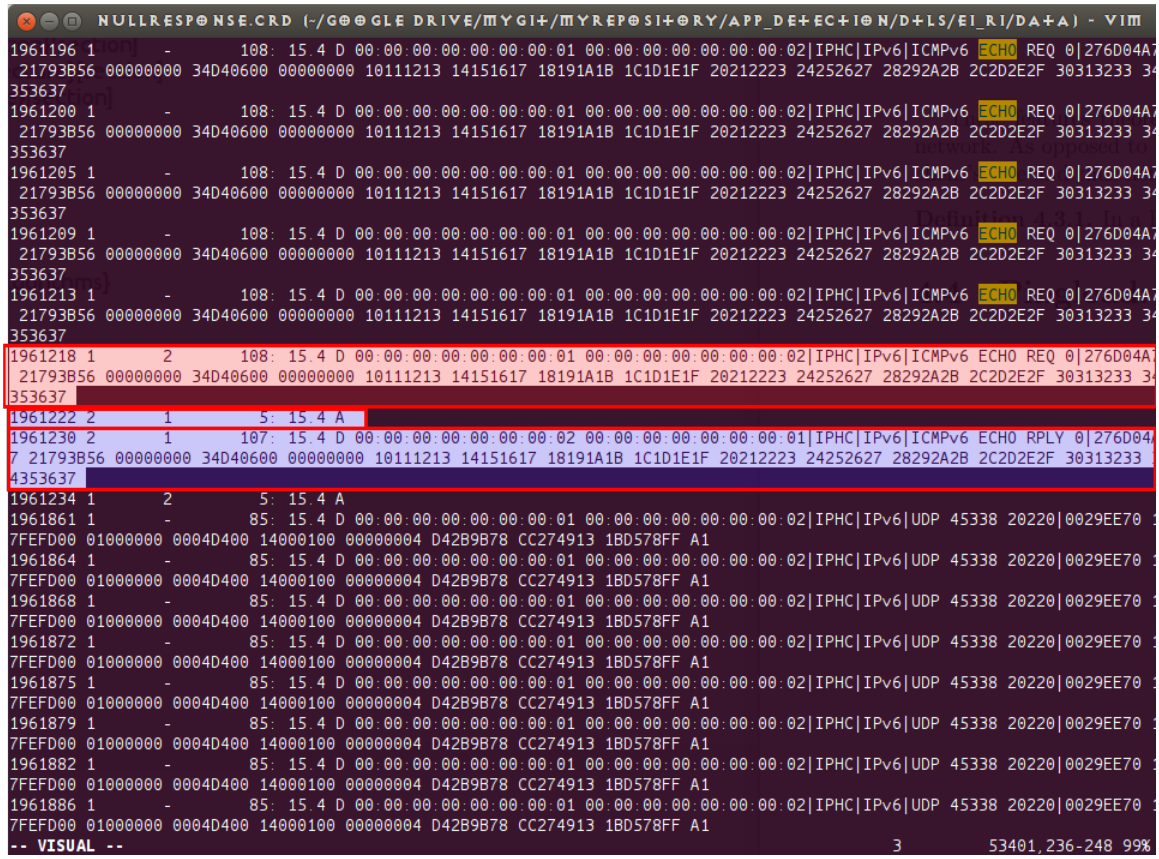

Figure 4.1: Capture of a ping packet

the client's side and thus network latency (RTT[3]) must be concerned, being able to capture all traffic in the network provides much more accurate timing information and hence may be exploited to develop more efficient attacks.

Definition 4.3.1. In a Request-Response application model, **RI**, **Response Interval**, is defined as the interval between a request packet being received and its response being sent.

Example 4.3.1. Three packets are marked out in Figure 4.1 which forms an instance of ICMP ECHO[8] (also known as PING) session. The extracted packet features are displayed in Table 4.1.

Explanation of the Packets:

The first packet is an ICMP ECHO request and the third packet being its

[3]Round-Trip Time

Time (ms)	From (ID)	To (ID)	Length (bytes)	Type
1961218	1	2	108	ICMP ECHO
1961222	2	1	5	802.15.4 ACK
1961230	2	1	107	ICMP ECHO

Table 4.1: Packet Features of an ICMP ECHO request and response

response. The second packet is a 802.15.4 ACK[4] and is thus transparent to the upper ICMP protocol.

From this example we can see that the RI for this PING session is:

$$1961230 - 1961218 = 12(\text{ms})$$

Timing information can be exploited by several attacks, such as [9] and [10].

We have experimentally measured a RNG[5] call on Wismote platform in the Cooja simulator is roughly 0.3 ms.

4.4 PINGLOAD: PING side-channel for Payload

Support of ICMP ECHO is required by [8] and is also enabled in Contiki OS by default. However, our experimental results shows that the response time of these ping packets could potentially be exploited to reveal the application running on target sensor node.

We call such technique **Application Fingerprinting**.

4.4.1 Hypothesis

A phenomenon we realised is that when a ping packet arrived while the target node is executing some payload, say reading a sensor or processing data, the PING RI begins to vary comparing to a stable value when no there is no payload.

Example 4.4.1. Figure 4.2 shows RIs of PING collected in two experiments. The upper half are collected with the target is constantly sleep whilst the lower half occasionally receives a request which triggers the target to call RNG. We can see that the PING RI varies alongside the target is given some payload from this figure.

[4]This is an acknowledgement from the receiver that notifies the sender that the packet has been received.

[5]Random Number Generator

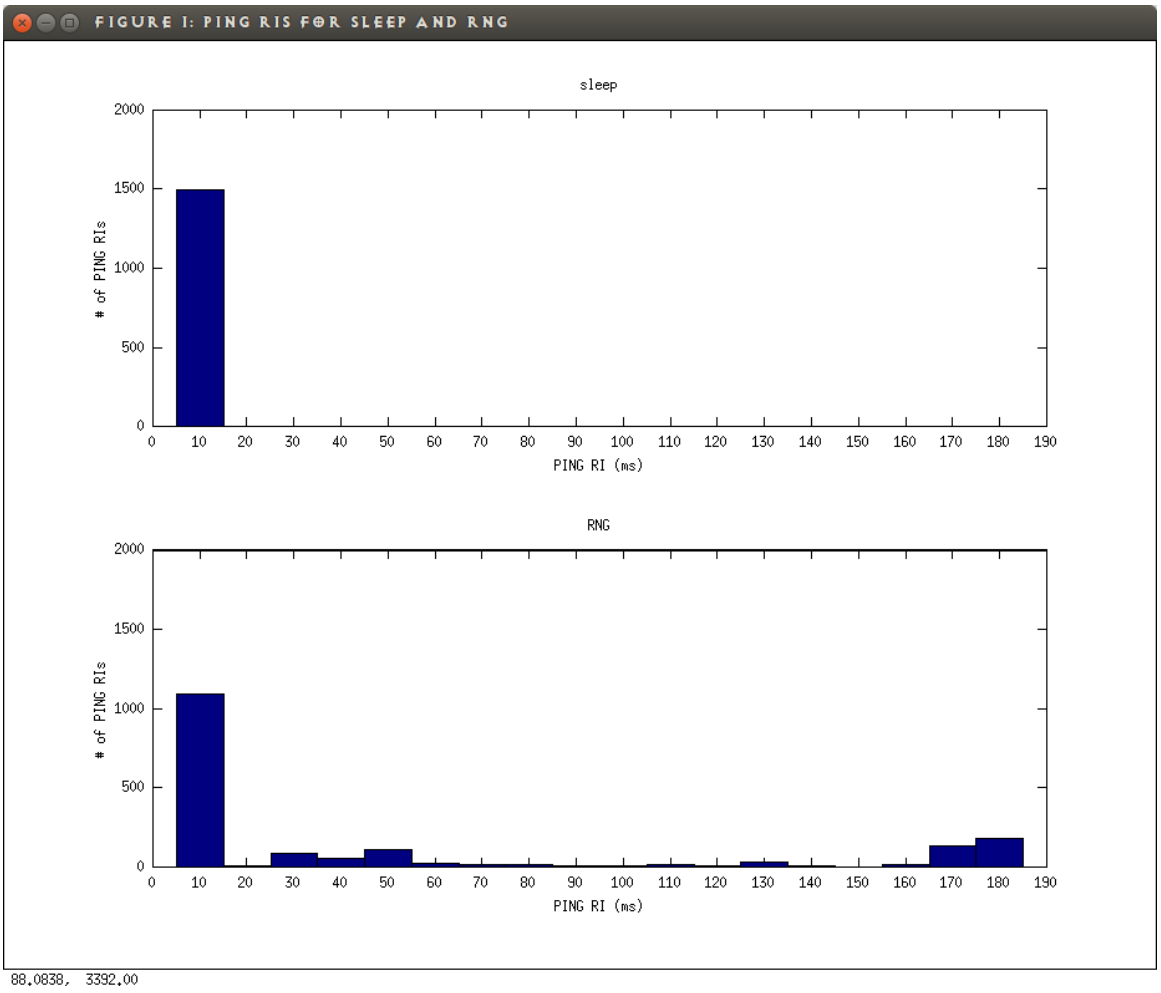

Figure 4.2: An example of PING RIs with different payload

The data shown in Figure 4.2 suggests that the "plain", that is without any interference, PING RI is 12 to 13 ms. Further more, those variations of PING RI is very likely caused by the payload of the target.

This experiment inspired that the distributions of PING RIs might vary according to the payload of target and could possibly considered as an fingerprint of the target's application. In other word, an adversary could possibly tell whether the target is running a specific application by looking at its PING RIs distribution.

The attack is strait forward:

Profile sleep RIs :
The PING RIs for a sleeping node of the same platform can be profiled by pinging a sleeping node. We denote the sleeping profile as RI_{sleep}.

Fingerprint application :
The adversary collects PING RIs on a profiling node with known application. The profiling node needs to be of the same platform and executing the same code of the target's. The application fingerprint denotes as: $F_p = \{p_1, p_2, ..., p_n\}$.

Collect fingerprint of target :
The adversary then collects the PING RI for the target node by pinging it. We denote the collected data as: $F_t = \{t_1, t_2, ..., t_m\}$.

Extract Featured RIs : We can remove them from the data sets and keep the PING RIs those has been interfered by the application. We denote the extracted RIs as **Featured RIs**:

$$F'_p = \{x | x \in F_p, x \notin RI_{sleep}\}$$
$$F'_p = \{x | x \in F_t, x \notin RI_{sleep}\}$$

Practically speaking, the PING protocol are designed to be responded immediately for diagnosis purpose; hence RI_{sleep} usually has an extremely low variance and its mean is also much less than F_p and F_t.

Using the Featured RIs not only provides a better vision of the fingerprint but also removes the error caused by different frequency of the target code being executed, as all the Featured RIs are actually collected when the node is at a non-sleeping state.

Estimate Distribution (Optional) :
We then estimate the distributions of F'_p and F'_t, denote as D_p and D_t. A naive method is to simply use their histograms. An example of such histograms are shown as Figure 4.3.

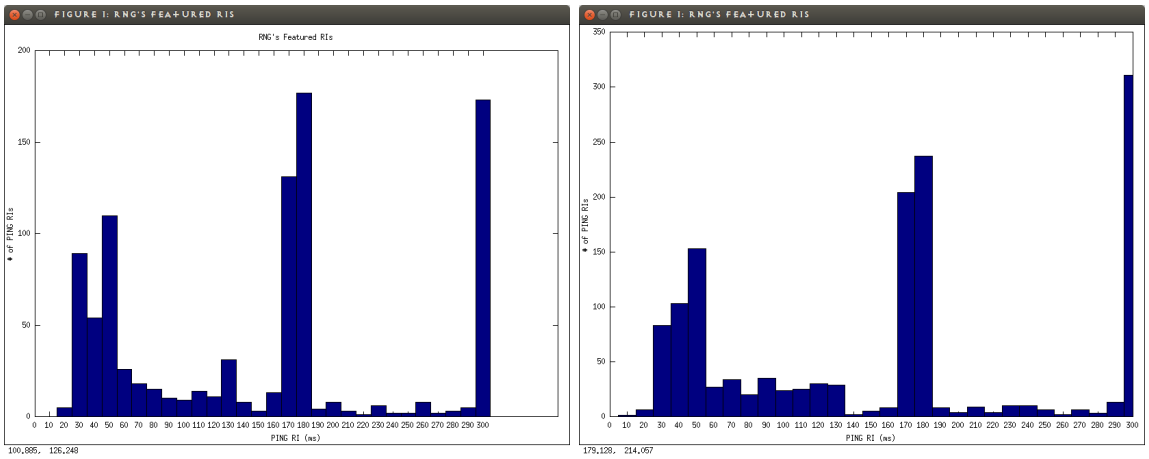

Figure 4.3: Two examples of RNG's Featured RIs histogram

Distinguish Distributions :

Finally we test whether D_t and D_p are the same distribution. An naive way is to compute the correlation of counts of the histograms. We conclude the target node is running the profiled application if D_t and D_p are the same distribution.

Practically speaking, the key point of Application Fingerprinting is to test whether the target's Featured RIs, i.e. F'_t, is sampled from same distribution of the profiled one, i.e. F'_p; therefore estimating their distribution might not be necessary for some statistical methods such as t-tests. However as we can see in Figure 4.3, PING RIs' distribution are very unlikely to be normalised. Therefore a future work is to find a better distinguishing method than the current naive one.

4.4.2 Experiment Result

We tried our Application Fingerprinting method above on a Cooja simulated Wismote platform with different code. **In conclusion, the fingerprint appears to be effective for certain circumstances but will tends to result into false positives as the profiled application and target application gets similar to each other.**

To be more specifically, the target node execute some specific code upon receiving an application layer protocol request, similar to CoAP. Further more, all traffic are protected by DTLS with TLS_PSK_WITH_AES_128_CCM_8 ciphersuite. Both intervals of PINGs and the application request are set to some asynchronised value to avoid overflooding the target and to create a 'more realistic' simulation.

Everything other than the examined code are the same for all experiments. Two

Correlations	i=50	i=100	i=2500	i=5000
i=50	**0.988**	0.891	-0.014	-0.033
i=100	0.891	**0.973**	-0.025	-0.042
i=2500	-0.014	-0.025	**0.993**	-0.035
i=5000	-0.033	-0.042	-0.035	**0.985**

Table 4.2: Correlations for RNG

Correlations	+	*	%
+	**0.990**	**0.990**	**0.988**
*	**0.990**	**0.989**	**0.985**
%	**0.988**	**0.985**	**0.984**

Table 4.3: Correlations for word arithmetic operations

samples are collected independently for each code to simulate a fingerprinting scenario. The histograms are clustered by 5ms.

We examined two classes of codes:

RNG Calls :

The target node repeatedly calls RNG for i times. We examined their Featured PING RI for different values of i. The reason for picking RNG is that on some platforms where a hardware RNG is provided, the call to it is expected to be similar to a call to a sensor reading which is actually an interrupt. Results are shown in Table 4.2.

Arithmetic Operations :

The target node repeatedly does arithmetic operations, namely addition, multiplication and modular, on two random generated word size integers for 10000 times. This class is particularly interested from a cryptographic point of view as the number of arithmetic operations could potentially developed to key recovering attacks. Results are shown in Table 4.3.

We also computed the correlation for $i = 50$ and addition, as shown in Table 4.4. The results suggests the following conjectures:

Correlations	+
i=50	0.877

Table 4.4: Correlation for $i = 50$ and addition

1. The results for RNG suggests that the fingerprinting is effective for this class of code, as the same code results into nearly perfect correlations (≥ 0.95).

2. Even relatively slight changes can be detected, as we can see the correlation dropped to 0.891 alongside 50 iterations of RNG calls (50 RNG calls take about 1.4ms).

3. The results for arithmetic operations indicates that their fingerprint are unlikely to be distinguishable. There are two potential causes we have considered:

   (a) The differences between these operations are too small to be detected.

   (b) Experiment methodology error. Since the target node we used during the experiments call RNG twice upon each request to generate two operands whilst the word arithmetic operations have much lighter weigh comparing to RNG at magnitude level; thus the fingerprint is dominated by RNG rather than word arithmetic operations. As a result, we can see that a relatively high correlation can be observed between word addition and 50 RNG calls as shown in Table 4.4.

4.4.3 A General Hypothetical Model: Black Box Model

Although the experiments supports the hypothesis that the variation of PING RIs is relevant to the application, it is not yet clear which factor exactly caused the variations.

Therefore we suggest that this phenomenon is a result of multi factors instead, including:

- Code being executed and whether it is preemptive or non preemptive,
- Memory usage, such as packet buffers, or
- Any other hardware/software conditions.

It is difficult to verify these factors as many of them requires strict synchronisation between devices and the PING RIs are only showing statistical features.

Therefore we propose the Black Box Model which we hope could support further research.

Definition 4.4.1. The **Black Box Model** models the target device as a stateful black box. A **state**, denotes as $\vec{S} =< \text{factor}_1, \text{factor}_2 ... >$, is an abstracted vector of multiple factors that could affect PING RIs, such as current code context or memory usage. $\mathbb{D}_{\vec{S}}$ is the distribution of PING RIs when the target node is at state $\vec{S}$.

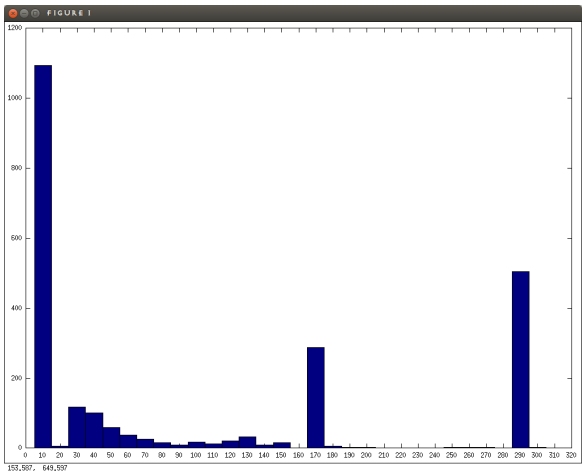

Figure 4.4: PING RIs of $\vec{S}_{main}$

We can redefine a state by multiple mutually exclusive substates, $\vec{S}_1$, $\vec{S}_2$ etc. Each sub-state has its own PING RI distribution $\mathbb{D}_{\vec{S}_1}$, $\mathbb{D}_{\vec{S}_2}$ etc. The redefinition process can further more be done recursively.

Notice that since each substate are mutually exclusive; therefore $\mathbb{D}_{\vec{S}}$ is a linear combination of all distributions associated with all the substates of $\vec{S}$, since:

$$\forall x : Prob(RI = x | \vec{s} = \vec{S}) = \sum_{i=0}^{n} \left(p_i * Prob(RI = x | \vec{s} = \vec{S}_i) \right)$$

where $\vec{s}$ is the state at any moment, p_i is $Prob(\vec{s} = \vec{S}_i)$ and n is the number of substates of $\vec{S}$.

Hence we can write

$$\mathbb{D}_{\vec{S}} = \sum_{i}^{n} \left(p_i * \mathbb{D}_{\vec{S}_i} \right) \tag{4.1}$$

Example 4.4.2. Take one of the RNG applications with $i = 5000$ in Table 4.2 for example.

We define:

$$\vec{S}_{main} =< \text{App} = \text{RNG}, i = 5000 >$$

Its associated PING RI distribution $\mathbb{D}_{\vec{S}_{main}}$ is approximated from the histogram shown in Figure 4.4.

We then redefine $\vec{S}_{main}$ to two exclusive sub-states depends on whether the target

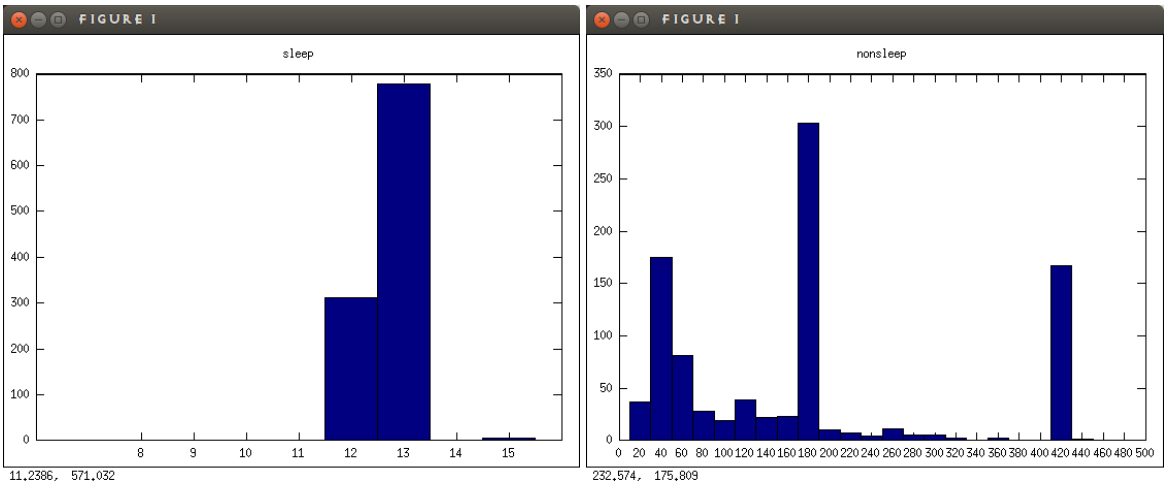

Figure 4.5: PING RIs of $\vec{S}_{sleep}$ and $\vec{S}_{nonsleep}$

is in a sleeping mode.

$$\begin{aligned} \vec{S}_{main} &= \vec{S}_{sleep} + \vec{S}_{non-sleep} \\ \vec{S}_{sleep} &=< \text{App} = \text{RNG}, i = 5000, \text{Code} = \text{sleep} > \\ \vec{S}_{non-sleep} &=< \text{App} = \text{RNG}, i = 5000, \text{Code} \neq \text{sleep} > \end{aligned}$$

Since we know by experiment that the PING RIs are 12 to 13ms when the target is sleep; therefore we can actually approximate $\mathbb{D}_{\vec{S}_{sleep}}$ by filter out those RIs in $[12, 13]$. Vice versa, the distribution of Featured RIs can be viewed as $\mathbb{D}_{\vec{S}_{nonsleep}}$ since they are exactly the RIs filtered by the RIs of sleep. The histograms are shown in Figure 4.5.

So we have

$$\mathbb{D}_{\vec{S}_{main}} = p_0 \mathbb{D}_{\vec{S}_{sleep}} + p_1 \mathbb{D}_{\vec{S}_{nonsleep}} \tag{4.2}$$

where p_0 and p_1 corresponds to the probability of the target being sleep and nonsleep.

Solving Equation (4.2) gives us roughly $p_0 = 0.538$ and $p_1 = 0.462$. Assuming the PING packets are received by the target randomly, we can estimate that the target is in a sleep for 53.8% of the time and awaken for 46.2%.

Theoretically we can further redefine more sub-states, e.g.:

$$\begin{aligned} \vec{S}_{nonsleep} &= \vec{S}_{RNG} + \vec{S}_{header} \\ \vec{S}_{RNG} &=< \text{App} = \text{RNG}, i = 5000, \text{Code} = \text{random\_rand}() > \\ \vec{S}_{header} &=< \text{App} = \text{RNG}, i = 5000, \text{Code} = \text{header processing} > \end{aligned}$$

However, obtaining their corresponded distributions poses a problem to this method, we leave this topic to future research.

Practically speaking, p_i might be an interesting information to an adversary as it reveals the internal state of the target which could lead to many information breach. In practice, we might only want to approximate the solutions of p_i, which effectively reduces the problem of approximating Equation (4.1) to a knapsack problem[11].

In most cases obtaining $\mathbb{D}_{\vec{S}_i}$ is difficult. We leave this problem to future research.

Chapter 5

Conclusion

In Chapter 2 we described a LLSEC implementation in Contiki OS *noncoresec* and argues that its selection of IV is not secure enough and potentially has a reset problem.

Chapter 3 discusses some implementation issues of DTLS on Contiki OS.

Chapter 4 first argues that under some natures of WSN, an adversary could possibly collect more accurate timing information than usual Internet Web-application attacker. Then we described a potential sensor node application fingerprinting attack using the interaction of PING protocol and the application running on the target node.

Chapter 6

Fulture Work

1. All the results are based on Contiki OS with 6LowPAN. However, Zigbee[12] protocol is not supported by Contiki OS. Considering the popularity and market share, other OSes and Zigbee should also be studied.

2. DTLS using TLS_ECHDE_ECDSA_WITH_AES_128_CCM_8 ciphersuite is ignored due to performance issue. However, it should be reconsidered in the future as it provides better security. Further more, implementation other than tinyDTLS, or protocols other than DTLS should also be examined.

3. The security brought by *noncoresec* is not satisfying. A solution is expected, such as better key exchange algorithms, ciphersuites, or choice of IV.

4. Some RPL messages use specific MAC header flag which can be seen even with LLSEC enabled. These flags are yet to be studied in this paper.

5. RNG is being used for convenience for simulating sensor readings in a simulator. It should be replaced by real sensor readings with real devices.

6. Timing informations are all collected using Cooja simulator. The experiments should be re done with real devices.

7. In Section 4.4.1, we used a naive correlation test to test the fingerprints. This method might be improved by using nonparametric tests such as Kolmogorov-Smirnov test.

Bibliography

[1] Naveen Sastry and David Wagner. "Security Considerations for IEEE 802.15.4 Networks". In: *Proceedings of the 3rd ACM Workshop on Wireless Security*. WiSe '04. Philadelphia, PA, USA: ACM, 2004, pp. 32–42. ISBN: 1-58113-925-X. DOI: 10.1145/1023646.1023654. URL: http://doi.acm.org/10.1145/1023646.1023654.

[2] URL: https://github.com/kkrentz/contiki/wiki.

[3] URL: http://sourceforge.net/projects/tinydtls/.

[4] *IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks-Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. Tech. rep. 2006, 0_1–305. DOI: 10.1109/ieeestd.2006.232110. URL: http://dx.doi.org/10.1109/ieeestd.2006.232110.

[5] G. Montenegro et al. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944 (Proposed Standard). Updated by RFCs 6282, 6775. Internet Engineering Task Force, Sept. 2007. URL: http://www.ietf.org/rfc/rfc4944.txt.

[6] Adam Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. 2011.

[7] Carsten Bormann Lars Schmertmann Klaus Hartke. *CoDTLS: DTLS handshakes over CoAP*. 2015. URL: https://tools.ietf.org/html/draft-schmertmann-dice-codtls-01.

[8] R. Braden. *Requirements for Internet Hosts - Communication Layers*. RFC 1122 (INTERNET STANDARD). Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. Internet Engineering Task Force, Oct. 1989. URL: http://www.ietf.org/rfc/rfc1122.txt.

[9] K.P. Dyer et al. "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail". In: *Security and Privacy (SP), 2012 IEEE Symposium on*. May 2012, pp. 332–346. DOI: 10.1109/SP.2012.28.

[10] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, pp. 104–113. ISBN: 3-540-61512-1. URL: http://dl.acm.org/citation.cfm?id=646761.706156.

[11] URL: https://en.wikipedia.org/wiki/Knapsack_problem.

[12] ZigBee Alliance. *ZigBee specification*. Tech. rep. June 2005.