6TiSCH Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2019

M. Tiloca
RISE AB
S. Duquennoy
Yanzi Networks AB
G. Dini
University of Pisa

June 10, 2019

Robust Scheduling against Selective Jamming in 6TiSCH Networks
draft-tiloca-6tisch-robust-scheduling-02

Abstract

This document defines a method to generate robust TSCH schedules in a 6TiSCH (IPv6 over the TSCH mode of IEEE 802.15.4-2015) network, so as to protect network nodes against selective jamming attack. Network nodes independently compute the new schedule at each slotframe, by altering the one originally available from 6top or alternative protocols, while preserving a consistent and collision-free communication pattern. This method can be added on top of the minimal security framework for 6TiSCH.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 12, 2019.

Table of Contents

1. Introduction

Nodes in a 6TiSCH network communicate using the IEEE 802.15.4-2015 standard and its Timeslotted Channel Hopping (TSCH) mode. Some properties of TSCH make schedule units, i.e. cells, and their usage predictable, even if security services are used at the MAC layer.

This allows an external adversary to easily derive the communication pattern of a victim node. After that, the adversary can perform a selective jamming attack, by covertly, efficiently, and effectively transmitting over the only exact cell(s) in the victim's schedule. For example, this enables the adversary to jeopardize a competitor's network, while still permitting their own network to operate correctly.

This document describes a method to counteract such an attack. At each slotframe, every node autonomously computes a TSCH schedule, as a pseudo-random permutation of the one originally available from 6top [RFC8480] or alternative protocols.

The resulting schedule is provided to TSCH and used to communicate during the next slotframe. In particular, the new communication pattern results unpredictable for an external adversary. Besides, since all nodes compute the same pseudo-random permutation, the new communication pattern remains consistent and collision-free.

The proposed solution is intended to operate on slotframes that are used for data transmission by current network nodes, and that are not used to join the network. In fact, since the TSCH schedule is altered at each slotframe, the proposed method cannot be applied to slotframes that include a "minimal cell" [RFC8180] and possible other randez-vouz cells used for joining the 6TiSCH network.

This document specifies also how this method can be added on top of the minimal security framework for 6TiSCH and its Constrained Join Protocol (CoJP) [I-D.ietf-6tisch-minimal-security].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with terms and concepts defined in [I-D.ietf-6tisch-minimal-security], [I-D.ietf-6tisch-terminology] and [RFC8152].

This document refers also to the following terminology.

- Permutation key. A cryptographic key shared by network nodes and used to permute schedules. Different keys are used to permute the utilization pattern of timeslots and of channelOffsets.

2. Properties of TSCH that Simplify Selective Jamming

This section highlights a number of properties of the TSCH cell usage that greatly simplify the performance of the selective jamming attack described in Section 3.

Given:

- N_S as the size of slotframes in timeslots;
- N_C as the number of available channelOffsets;
- The channel 'f' to communicate at timeslot 's' with ASN and channelOffset 'chOff' computed as f = F[(ASN + chOff) mod N_C];

And assuming for simplicity that:

- N_S and N_C are coprime values;
- The channel hopping sequence is N_C in size and equal to {0, 1, ..., N_C - 1};

Then, the following properties hold:

- Periodicity property. The sequence of channels used for communication by a certain cell repeats with period (N_C x N_S) timeslots.
- Usage property. Within a period, every cell uses all the available channels, each of which only once.
- Offset property. All cells follow the same sequence of channels with a certain offset.
- Predictability property. For each cell, the sequence of channels is predictable. That is, by knowing the channel used by a cell in a given timeslot, it is possible to compute the remaining channel hopping sub-sequence.

  In fact, given a cell active on channel 'f' and timeslot 's' on slotframe 'T', and since ASN = (s + T x N_S), it holds that

  f = [(s + T x N_S + chOff) mod N_C] (Equation 1)

  By solving this equation in 'chOff', one can predict the channels used by the cell in the next sloframes. Note that, in order to do that, one does not need to know the absolute number 'T' of the slotframe (and thus the exact ASN) in which timeslot 's' uses a

certain channel 'f'. In fact, one can re-number slotframes starting from any arbitrarily assumed "starting-slotframe".

3. Selective Jamming Attack

This section describes how an adversary can exploit the properties listed in Section 2, and determine the full schedule of a victim node, even if security services at the MAC layer are used.

This allows the adversary to selectively jam only the exact cell(s) in the victim's schedule, while greatly limiting the exposure to detection. At the same time, the attack is highly effective in jeopardizing victim's communications, and is highly energy-efficient, i.e., can be carried out on battery.

For simplicity, the following description also assumes that a victim node actually transmits/receives during all its allocated cells at each slotframe.

3.1. Adversary Model

This specification addresses an adversary with the following properties.

- The adversary is external, i.e. it does not control any node registered in the 6TiSCH network.
- The adversary wants to target precise network nodes and their traffic. That is, it does not target the 6TiSCH network as a whole, and does not perform a wide-band constant jamming.
- The adversary is able to target multiple victim nodes at the same time. This may require multiple jamming sources and/or multiple antennas per jamming source to carry out the attack.

Furthermore, compared to wide-band constant jamming, the considered selective jamming attack deserves special attention to be addressed, due to the following reasons.

- It is much more energy efficient.
- It minimizes the adversary's exposure and hence the chances to be detected.
- It has the same effectiveness on the intended victim nodes. That is, it achieves the same goal, while avoiding the unnecessarily exposure and costs of wide-band constant jamming.

It is worth noting that a wide-band constant jamming can achieve the same result more easily, in the extreme cases where the target slotframe is (nearly) fully used by a few nodes only, or the adversary has as many antennas as the number of available channels. However, this would still come at the cost of high exposure and higher energy consumption for the adversary.

3.2. Attack Example

The following example considers Figure 1, where N_S = 3, N_C = 4, and the channel hopping sequence is {0,1,2,3}. The shown schedule refers to a network node that uses three cells 'L_1', 'L_2' and 'L_3', with {0,3}, {1,1} and {2,0} as pairs {timeslot, channelOffset}, respectively.

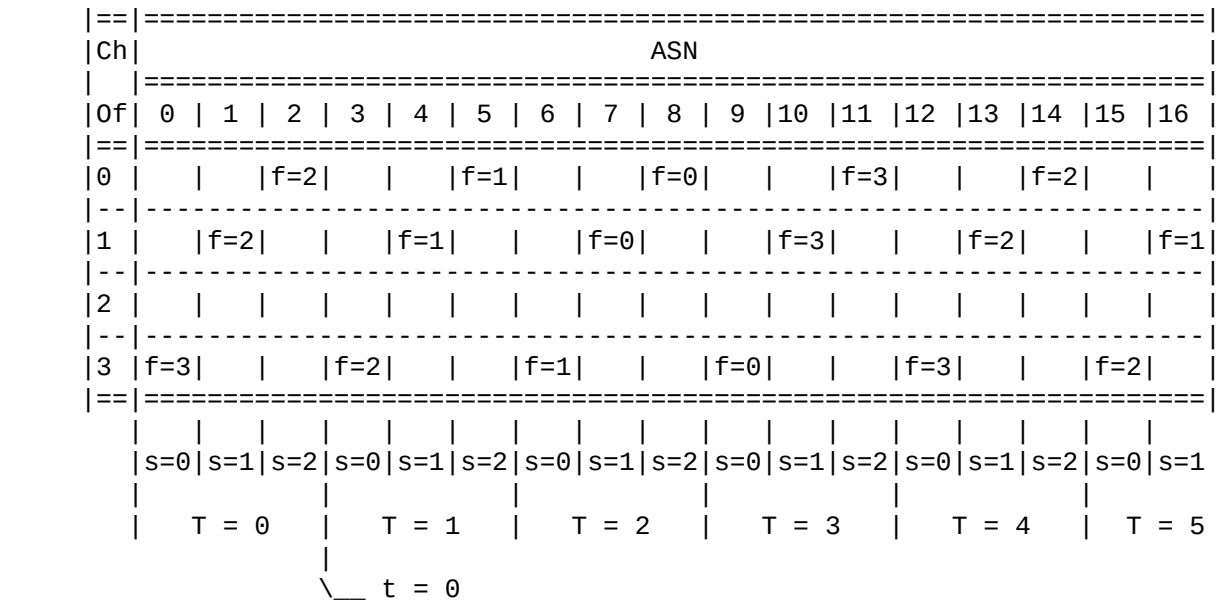

Figure 1: Attack Example with Slotframe Re-numbering

1. The adversary starts the attack at absolute slotframe T = 1, which is assumed as "starting-slotframe" and thus renamed as slotframe t = 0. The renaming is possible due to the offset and predictability properties.

2. The adversary picks a channel 'f*' at random, and monitors it for N_C consecutive slotframes to determine the timeslots in which the victim node communicates on that channel. Due to the usage property, the number of such timeslots is equal to the number of cells assigned to the victim node.

With reference to Figure 1, if, for example, f* = 1, the adversary determines that the victim node uses channel 'f*' in timeslots s = 1 and s = 2 of slotframe t = 0 and in timeslot s = 0 of slotframe t = 1. The adversary can then deduce that the victim node uses three different cells 'L_1', 'L_2' and 'L_3', in timeslots 0, 1 and 2, respectively.

3. The adversary determines the channels on which the victim node is going to transmit in the next slotframes, by exploiting the predictability property.

   That is, by instantiating Equation 1 for cell L_1, timeslot s = 0 and slotframe t = 1, one gets [1 = (3 + chOff_1) mod 4], which has solution for chOff_1 = 2. Hence, the function to predict the channel 'f_1' to be used by cell 'L_1' in a slotframe 't', t >= 1, is f_1 = [(2 + 3 x t) mod 4], which produces the correct periodic sequence of channels {1, 0, 3, 2}. Similarly, one can instantiate Equation 1 for cells 'L_2' and 'L_3', so producing the respective periodic sequence of channels {1,0,3,2} and {1,0,3,2}.

4. The adversary has discovered the full schedule of the victim node and can proceed with the actual selective jamming attack. That is, according to the found schedule, the adversary transmits over the exact cells used by the victim node for transmission/ reception, while staying quiet and saving energy otherwise. This results in a highly effective, highly efficient and hard to detect attack against communications of network nodes.

4. Building Robust Schedules

This section defines a method to protect network nodes against the selective jamming attack described in Section 3. The proposed method alters the communication pattern of all network nodes at every slotframe, in a way unpredictable for the adversary.

At each slotframe 'T', network nodes autonomously compute the communication pattern for the next slotframe 'T+1' as a pseudo-random permutation of the one originally available. In order to ensure that the new communication pattern remains consistent and collision-free, all nodes compute the same permutation of the original one. In particular, at every slotframe, each node separately and independently permutes its timeslot utilization pattern (optionally) as well as its channelOffset utilization pattern.

To perform the required permutations, all network nodes rely on a same secure pseudo-random number generator (SPRNG) as shown in Figure 2, where E(x,y) denotes a cipher which encrypts a plaintext

'y' by means of a key 'x'. Network nodes MUST support the AES-CCM-16-64-128 algorithm.

```
unsigned random(unsigned K, unsigned z) {
    unsigned val = E(K,z);
    return val;
}
```

Figure 2: Secure Pseudo-Random Number Generator

All network nodes share the same following pieces of information.

- K_s, a permutation key used to permute the timeslot utilization pattern, and used as input to the random() function in Figure 2. K_s is provided upon joining the network, and MAY be provided as described in Section 5.

- K_c, a permutation key used to permute the channelOffset utilization pattern, and used as input to the random() function in Figure 2. K_c is provided upon joining the network, and MAY be provided as described in Section 5.

- z_s, a counter used to permute the timeslot utilization pattern, and used as input to the random() function in Figure 2. At the beginning of each slotframe, z_s is equal to [(N_S - 1) x floor(ASN* / N_S)], where ASN* is the ASN value of the first timeslot of that slotframe. Then, z_s grows by (N_S - 1) from the beginning of a slotframe to the beginning of the next one.

- z_c, a counter used to permute the channelOffset utilization pattern, and used as input to the random() function in Figure 2. At the beginning of each slotframe, z_c is equal to [(N_C - 1) x floor(ASN* / N_S)], where ASN* is the ASN value of the first timeslot of that slotframe. Then, z_c grows by (N_C - 1) from the beginning of a slotframe to the beginning of the next one.

Then, at every slotframe, each network node takes the following steps, and generates its own permuted communication schedule to be used at the following slotframe. The actual permutation of cells relies on the well-known Fisher-Yates algorithm, that requires to generate (n - 1) pseudo-random numbers in order to pseudo-randomly shuffle a vector of n elements.

1. First, a pseudo-random permutation is performed on the timeslot dimension of the slotframe. This requires (N_S - 1) invocations of random(K,z), consistently with the Fisher-Yates algorithm. In particular, K = K_s, while z_s is passed as second argument and is incremented by 1 after each invocation. The result of this

step is a permuted timeslot utilization pattern, while the channelOffset utilization pattern is not permuted yet.

2. Second, a pseudo-random permutation is performed on the channelOffset dimension of the slotframe. This requires (N_C - 1) invocations of random(K,z), consistently with the Fisher-Yates algorithm. In particular, K = K_c, while z_c is passed as second argument and is incremented by 1 after each invocation. The result of this step is a fully shuffled communication pattern.

The resulting schedule is then provided to TSCH and considered for sending/receiving traffic during the next slotframe.

As further discussed in Section 6.3, it is possible to skip step 1 above, and hence permute only the channelOffset utilization pattern, while keeping a static timeslot utilization pattern.

Note for implementation: the process described above can be practically implemented by using two vectors, i.e. one for shuffling the timeslot utilization pattern and one for shuffling the channelOffset utilization pattern.

5. Adaptation to the 6TiSCH Minimal Security Framework

The security mechanism described in this specification can be added on top of the minimal security framework for 6TiSCH [I-D.ietf-6tisch-minimal-security].

That is, the two permutation keys K_s and K_c can be provided to a pledge when performing the Constrained Join Protocol (CoJP) defined in Section 8 of [I-D.ietf-6tisch-minimal-security].

To this end, the Configuration CBOR object [RFC7049] used as payload of the Join Response Message and defined in Section 8.4.2 of [I-D.ietf-6tisch-minimal-security] is extended with two new CoJP parameters defined in this specification, namely 'permutation key set' and 'permutation cipher'. The resulting payload of the Join Response message is as follows.

```
Configuration = {
   ? 2   : [ +Link_Layer_Key ],     ; link-layer key set
   ? 3   : Short_Identifier,        ; short identifier
   ? 4   : bstr,                    ; JRC address
   ? 6   : [ *bstr ],               ; blacklist
   ? 7   : uint,                    ; join rate
   ? TBD : [ +Permutation_Key ],    ; permutation key set
   ? TBD : Permutation_Cipher       ; permutation cipher
}
```

The parameter 'permutation key set' is an array encompassing one or two permutation keys encoded as byte strings. That is, the encoding of each individual permutation key is as follows.

```
Permutation_Key = (
      key_value          : bstr
(
```

If the 6TiSCH network uses the security mechanism described in this specification, the parameter 'permutation key set' MUST be included in the CoJP Join Response message and the pledge MUST interpret it as follows.

- In case only one permutation key is present, it is used as K_c to permute the channelOffset utilization pattern, as per Section 4.

- In case two permutation keys are present, the first one is used as K_s to permute the timeslot utilization pattern, while the second one is used as K_c to permute the channelOffset utilization pattern, as per Section 4. The two keys MUST have the same length.

The parameter 'permutation cipher' indicates the encryption algorithm used for the secure pseudo-random number generator as per Figure 2 in Section 4. The value is one of the encryption algorithms defined for COSE [RFC8152], and is taken from Tables 9, 10 and 11 of [RFC8152]. In case the parameter is omitted, the default value of AES-CCM-16-64-128 (COSE algorithm encoding: 10) MUST be assumed.

5.1. Error Handling

In case 'permutation key set' includes two permutation keys with different length or more than two permutation keys, the pledge considers 'permutation key set' not valid and MUST signal the error as specified in Section 8.3.1 of [I-D.ietf-6tisch-minimal-security].

The pledge MUST validate that keys included in 'permutation key set' are appropriate for the encryption algorithm specified in 'permutation cipher' or assumed as default. In case of failed validation, the pledge MUST signal the error as specified in Section 8.3.1 of [I-D.ietf-6tisch-minimal-security].

6. Security Considerations

With reference to Section 3.9 of [RFC7554], this specification achieves an additional "Secure Communication" objective, namely it defines a mechanism to build and enforce a TSCH schedule which is

robust against selective jamming attack, while at the same time consistent and collision-free.

Furthermore, the same security considerations from the minimal security framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] hold for this document. The rest of this section discusses a number of additional security considerations.

6.1. Effectiveness of Schedule Shuffling

The countermeasure defined in Section 4 practically makes each node's schedule look random to an external observer. Hence, it prevents the adversary from performing the attack described in Section 3.

Then, a still available strategy for the adversary is to jam a number of cells selected at random, possibly on a per-slotframe basis. This considerably reduces the attack effectiveness in successfully jeopardizing victims' communications.

At the same time, nodes using different cells than the intended victims' would experience an overall slightly higher fraction of corrupted messages. In fact, the communications of such accidental victims might be corrupted by the adversary, when they occur during a jammed timeslot and exactly over the channelOffset chosen at random.

6.2. Renewal of Key Material

It is RECOMMENDED that the two permutation keys K_s and K_c are revoked and renewed every time a node leaves the network. This prevents a leaving node to keep the permutation keys, which may be exploited to selectively jam communications in the network.

This rekeying operation is supposed to be performed anyway upon every change of network membership, in order to preserve backward and forward security. In particular, new IEEE 802.15.4 link-layer keys are expected to be distributed before a new pledge can join the network, or after one or more nodes have left the network.

The specific approach to renew the two permutation keys, possibly together with other security material, is out of the scope of this specification.

6.3. Static Timeslot Allocations

As mentioned in Section 4 and Section 5, it is possible to permute only the channelOffset utilization pattern, while preserving the originally scheduled timeslot utilization pattern. This can be desirable, or even unavoidable in some scenarios, in order to

guarantee end-to-end latencies in multi-hop networks, as per accordingly designed schedules.

However, preserving a static timeslot utilization pattern would considerably increase the attack surface for a random jammer adversary. That is, the adversary would immediately learn the timeslot utilization pattern of a victim node, and would have a chance to successfully jam a victim's cell equal to (1 / N_C).

6.4. Network Joining Through Randez-vous Cells

As described in [I-D.ietf-6tisch-minimal-security], a pledge joins a 6TiSCH network through a Join Proxy (JP), according to the Constrained Join Protocol (CoJP) and based on the information conveyed in broadcast Enhanced Beacons (EBs). In particular, the pledge will communicate with the JP over randez-vous cells indicated in the EBs.

In practice, such cells are commonly part of a separate slotframe, which includes one scheduled "minimal cell" [RFC8180], typically used also for broadcasting EBs. Such slotframe, i.e. Slotframe 0, usually differs from the slotframe(s) used for both EBs and data transmission.

In order to keep the join process feasible and deterministic, the solution described in this specification is not applied to Slotframe 0 or any other slotframes that include randez-vous cells for joining. As a consequence, an adversary remains able to selectively jam the "minimal cell" (or any randez-vous cell used for joining), so potentially jeopardizing the CoJP and preventing pledges to join the network altogether.

7. IANA Considerations

This document has the following actions for IANA.

7.1. Permutation Key Set

IANA is asked to enter the following value into the "Constrained Join Protocol Parameters Registry" defined in [I-D.ietf-6tisch-minimal-security] and within the "IPv6 over the TSCH mode of IEEE 802.15.4e (6TISCH) parameters" registry.

Name	Label	CBOR type	Description	Reference
permutation key set	TBD	array	Identifies the array including one or two permutation keys to alter cell utilization	[[this document]]

7.2. Permutation Cipher

IANA is asked to enter the following value into the "Constrained Join Protocol Parameters Registry" defined in [I-D.ietf-6tisch-minimal-security] and within the "IPv6 over the TSCH mode of IEEE 802.15.4e (6TISCH) parameters" registry.

Name	Label	CBOR type	Description	Reference
permutation cipher	TBD	integer	Identifies the cipher used for generating pseudo-random numbers to alter cell utilization	[[this document]]

8. References

8.1. Normative References

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-10 (work in progress), April 2019.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2. Informative References

[I-D.ietf-6tisch-terminology] Palattella, M., Thubert, P., Watteyne, T., and Q. Wang, "Terms Used in IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-terminology-10 (work in progress), March 2018.

[RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement", RFC 7554, DOI 10.17487/RFC7554, May 2015, <https://www.rfc-editor.org/info/rfc7554>.

[RFC8180] Vilajosana, X., Ed., Pister, K., and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration", BCP 210, RFC 8180, DOI 10.17487/RFC8180, May 2017, <https://www.rfc-editor.org/info/rfc8180>.

[RFC8480] Wang, Q., Ed., Vilajosana, X., and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)", RFC 8480, DOI 10.17487/RFC8480, November 2018, <https://www.rfc-editor.org/info/rfc8480>.

[Test-Implementation] "Test Implementation in C with OpenSSL", May 2019, <https://gitlab.com/crimson84/draft-tiloca-6tisch-robust-scheduling/tree/master/test>.

Appendix A. Test Vector

This appendix provides a test vector for an example where the method proposed in this document is used to generate robust TSCH schedules.

The example focuses on one network node and considers the schedule in Figure 1 as the original schedule to permute at each slotframe.

The results shown in this example have been produced using the implementation available at [Test-Implementation].

A.1. Detailed Technique

In this example, the permutation of the timeslot utilization pattern and of the channelOffset utilization pattern occurs as follows.

A.1.1. Data Structures and Schedule Encoding

Each network node maintains two vectors X_s and X_c, each composed of N_S unsigned integer values. At the beginning of each slotframe, X_s and X_c indicate the node's original schedule. In particular:

- X_s indicates the usage of timeslots in the slotframe. That is, the element X_s[i] refers to the i-th timeslot of the slotframe.

- X_c indicates the usage of channelOffsets at each timeslot in the slotframe. That is, X_c[i] refers to the channelOffset value used in the i-th timeslot of the slotframe.

Then, the two vectors encode the schedule information as follows:

- If the i-th timeslot is not used, X_s[i] = 0 and X_c[i] = N_C.

- If the i-th timeslot is used to transmit with channelOffset 'c', X_s[i] = 1 and X_c[i] = c.

- If the i-th timeslot is used to receive with channelOffset 'c', X_s[i] = 2 and X_c[i] = c.

Note that optimized implementations can achieve the same goal with permutation vectors of smaller size.

A.1.2. Pseudo-Random Number Generation

When invoking E() within the random() function in Figure 2:

- The second parameter has 5 bytes in size like the ASN, and is provided as plaintext to the permutation cipher.

- A copy of the second parameter is left-padded with 8 octects with value 0x00. The result is provided as 13-byte nonce to the permutation cipher, i.e. AES-CCM-16-64-128 in this example.

- No additional authenticated data are provided to the permutation cipher.

The unsigned value returned by E() and random() is the computed ciphertext left-padded with 3 octects with value 0x00. That is, the returned value is 8 bytes in size.

A.1.3. Array Permutation

To produce the required permutations, this example considers the Fisher-Yates modern version in Figure 3, which requires (n - 1) swaps to shuffle an array of n elements.

```
// Shuffle an array 'a' of 'n' elements (indices 0, ... , n - 1)
for i from (n - 1) down to 1 do {
   j = random integer such that 0 <= j <= i;
   exchange a[j] and a[i];
}
```

Figure 3: Fisher-Yates algorithm

At each step of the loop, 'j' is computed as r % (i + 1), where '%' is the modulo operator, and 'r' is the value returned by the function random() in Figure 2, as described in Appendix A.1.2.

A.1.4. Schedule Permutation

At each slotframe, the original schedule is considered as starting point to produce the permuted schedule for the following slotframe.

In particular, the permuted schedule for the following slotframe is computed according to the following steps.

1. The same pseudo-random permutation is performed on both vectors X_s and X_c, by using the Fisher-Yates algorithm in Figure 3. This requires (N_S - 1) invocations of random(K,z). In particular, K = K_s, while z_s is passed as second argument and is incremented by 1 after each invocation. As a result, X_s specifies the permuted timeslot utilization pattern, whereas X_c specifies a consistent while temporary channelOffset utilization pattern.

2. A vector Y of size N_C is produced, as a permutation of {0, 1, ..., N_C - 1} performed by using the Fisher-Yates algorithm in Figure 3. This requires (N_C - 1) invocations of random(K,z). In particular, K = K_c, while z_c is passed as second argument and is incremented by 1 after each invocation.

3. The vector X_c is updated as follows. Each element X_c[i] that refers to a non active timeslot, i.e. X_c[i] = N_C, is left as is. Otherwise, X_c[i] takes as value Y[j], where j = X_c[i].

As a result, the two permuted vectors X_s and X_c together provide a full communication pattern to use during the next slotframe.

A.2. Test Configuration

```
N_S = 3 // Slotframe size, in timeslots

N_C = 4 // Available channel offsets

Channel hopping sequence = {0, 1, 2, 3}

X_s = {1, 1, 2} // Original timeslot utilization pattern {Tx, Tx, Rx}

X_c = {3, 1, 0} // Original channelOffset utilization pattern

Starting ASN = 0

Permutation cipher: AES-CCM-16-64-128

K_s = { 0xce, 0xb0, 0x09, 0xae, 0xa4, 0x45, 0x44, 0x51,
        0xfe, 0xad, 0xf0, 0xe6, 0xb3, 0x6f, 0x45, 0x55 }

K_c = { 0xce, 0xb0, 0x09, 0xae, 0xa4, 0x45, 0x44, 0x51,
        0xfe, 0xad, 0xf0, 0xe6, 0xb3, 0x6f, 0x45, 0x56 }
```

A.3. Example Output

```
****************** ****************** ******************

START ROUND 1 of 2

The slotframe starts with: ASN = 0; z_s = 0; z_c = 0

****************** ****************** ******************

-- Start shuffling the time offsets --

---------- ---------- ----------

Counter (z_s): 0

Plaintext: 0x0000000000 (5 bytes)

Cipher nonce: 0x00000000000000000000000000 (13 bytes)

Ciphertext: 0xbedca72db3 (5 bytes)

Padded ciphertext: 0x000000bedca72db3 (8 bytes)

Fisher-Yates swap index i: 2
```

```
   Fisher-Yates swap-index j: 0
   ---------- ---------- ----------
   Counter (z_s): 1
   Plaintext: 0x0000000001 (5 bytes)
   Cipher nonce: 0x00000000000000000000000001 (13 bytes)
   Ciphertext: 0x23d36801f1 (5 bytes)
   Padded ciphertext: 0x00000023d36801f1 (8 bytes)
   Fisher-Yates swap index i: 1
   Fisher-Yates swap-index j: 1
   ---------- ---------- ----------
   -- Intermediate schedule --
   Timeslot utilization pattern X_s = {2, 1, 1}
   ChannelOffset utilization pattern X_c = {0, 1, 3}
   ---------- ---------- ----------
   -- Start shuffling the channel offset schedule --
   ---------- ---------- ----------
   Counter (z_c): 0
   Plaintext: 0x0000000000 (5 bytes)
   Cipher nonce: 0x00000000000000000000000000 (13 bytes)
   Ciphertext: 0x1e957fe44d (5 bytes)
   Padded ciphertext: 0x0000001e957fe44d (8 bytes)
   Fisher-Yates swap index i: 3
   Fisher-Yates swap-index j: 1
   ---------- ---------- ----------
```

Counter (z_c): 1

Plaintext: 0x0000000001 (5 bytes)

Cipher nonce: 0x00000000000000000000000001 (13 bytes)

Ciphertext: 0x6e2b990263 (5 bytes)

Padded ciphertext in bytes: 0x0000006e2b990263 (8 bytes)

Fisher-Yates swap index i: 2

Fisher-Yates swap-index j: 2

---------- ---------- ----------

Counter (z_c): 2

Plaintext: 0x0000000002 (5 bytes)

Cipher nonce: 0x00000000000000000000000002 (13 bytes)

Ciphertext: 0x4fae2cfe22 (5 bytes)

Padded ciphertext: 0x0000004fae2cfe22 (8 bytes)

Fisher-Yates swap index i: 1

Fisher-Yates swap-index j: 0

---------- ---------- ----------

Next slotframe starting with ASN = 3 will use:

- Shuffled timeslot schedule {2, 1, 1}, i.e. {Rx, Tx, Tx}.
- Shuffled channel offset schedule {3, 0, 1}.
- Shuffled frequencies schedule {2, 0, 2}.

********** ********** **********

START ROUND 2 OF 2

The slotframe starts with: ASN = 3; z_s = 2; z_c = 3

********** ********** **********

```
   -- Start shuffling the time offsets --
   ---------- ---------- ----------
   Counter (z_s): 2
   Plaintext: 0x0000000002 (5 bytes)
   Cipher nonce: 0x00000000000000000000000002 (13 bytes)
   Ciphertext: 0xd9a0c0f8eb (5 bytes)
   Padded ciphertext: 0x000000d9a0c0f8eb (8 bytes)
   Fisher-Yates swap index i: 2
   Fisher-Yates swap-index j: 2
   ---------- ---------- ----------
   Counter (z_s): 3
   Plaintext: 0x0000000003 (5 bytes)
   Cipher nonce: 0x00000000000000000000000003 (13 bytes)
   Ciphertext: 0x7aabd818ac (5 bytes)
   Padded ciphertext: 0x0000007aabd818ac (8 bytes)
   Fisher-Yates swap index i: 1
   Fisher-Yates swap-index j: 0
   ---------- ---------- ----------
   -- Intermediate schedules --
   Timeslot utilization pattern X_s = {1, 1, 2}
   ChannelOffset utilization pattern X_c = {1, 3, 0}
   ---------- ---------- ----------
   -- Start shuffling the channel offset schedule --
   ---------- ---------- ----------
```

```
   Counter (z_c): 3

   Plaintext: 0x0000000003 (5 bytes)

   Cipher nonce: 0x00000000000000000000000003 (13 bytes)

   Ciphertext: 0x947cf7c1d4 (5 bytes)

   Padded ciphertext: 0x000000947cf7c1d4 (8 bytes)

   Fisher-Yates swap index i: 3

   Fisher-Yates swap-index j: 0

   ---------- ---------- ----------

   Counter (z_c): 4

   Plaintext: 0x0000000004 (5 bytes)

   Cipher nonce: 0x00000000000000000000000004 (13 bytes)

   Ciphertext: 0xa9255744e7 (5 bytes)

   Padded ciphertext: 0x000000a9255744e7 (8 bytes)

   Fisher-Yates swap index i: 2

   Fisher-Yates swap-index j: 1

   ---------- ---------- ----------

   Counter (z_c): 5

   Plaintext: 0x0000000005 (5 bytes)

   Cipher nonce: 0x00000000000000000000000005 (13 bytes)

   Ciphertext: 0xa70a456e9e (5 bytes)

   Padded ciphertext: 0x000000a70a456e9e (8 bytes)

   Fisher-Yates swap index i: 1

   Fisher-Yates swap-index j: 0

   ---------- ---------- ----------
```

Next slotframe starting with ASN = 6 will use:

- Shuffled timeslot schedule {1, 1, 2}, i.e. {Tx, Tx, Rx}.
- Shuffled channel offset schedule {3, 0, 2}.
- Shuffled frequencies schedule {1, 3, 2}.

Acknowledgments

The authors sincerely thank Tengfei Chang, Michael Richardson, Yasuyuki Tanaka, Pascal Thubert and Malisa Vucinic for their comments and feedback.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE, and by the VINNOVA and Celtic-Next project CRITISEC.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Simon Duquennoy
Yanzi Networks AB
Isafjordsgatan 32C
Kista SE-16440 Stockholm
Sweden

Email: simon.duquennoy@yanzinetworks.com

Gianluca Dini
University of Pisa
Largo L. Lazzarino 2
Pisa 56122
Italy

Email: gianluca.dini@unipi.it