CHAOS: Exploiting Station Time Synchronization in 802.11 Networks

Sirus Shahini
University of Utah
sirus.shahini@utah.edu

Robert Ricci
University of Utah
ricci@cs.utah.edu

***Abstract*—Many locations, especially in urban areas, are quite noisy with WiFi traffic. In addition to data traffic, WiFi stations send management and control frames that can easily exceed several hundred frames per second just in one small area. These WiFi environments present the opportunity to transmit data through hiding it within the noise components that can be normal parts of benign transmissions. In this paper, we show how one particular feature of WiFi, the Timing Synchronization Function (TSF), can be exploited to create a fertile and robust channel for embedding secret signals. We take advantage of the fact that there is always some degree of imprecision reflected in time synchronization of WiFi stations.**

We present CHAOS, a new covert channel strategy to embed data bits in WiFi beacon frames using unmodified standard WiFi hardware. CHAOS makes use of the noise properties inherent in WiFi in two ways: First, it encodes information in the *ordering* of beacon frames, taking advantage of the fact that there is no natural or required ordering of beacons. Second, it makes use of a *timing channel* in the form of the TSF timestamp in management headers, imitating the natural imprecision of timing in real base stations to encode data in a way that is statistically similar to unmodified frames. CHAOS's parameters can be adjusted to configure data rate, the covert channel stability and frame miss rate; using our suggested settings, it is able to robustly broadcast secret data at 520 bits/s. We also show that TSF has substantial potential for further exploitation, sketching a correlation attack that uses it to map clients to base stations.

I. Introduction

Advancement of wireless technologies, coupled with the proliferation of Internet-based devices and the fast-growing demand for a constant and seamless connection to the Internet, has increased the global use of WiFi devices with an unprecedented speed. As of 2023, there were more than three times as many devices connected to IP networks as the population of the world [1]. There are an estimated 15 billion IoT devices that use WiFi connections on a daily basis [2], and this number is projected to at least double by 2030. The number of WiFi devices increased drastically after the implementation of the enhancements introduced in 802.11ac and 802.11ax standards due to the faster connections and better coverage, which make WiFi an appealing choice for data-intensive tasks and busy environments [3]. Furthermore, WiFi is provided not only through home access points (APs) but also it is normally offered as a complementary service in public places in urban areas, transit systems and modern private cars.

The magnitude of electronic devices that use wireless technologies, combined with their constant network usage, produces significant network traffic and consequently a tremendous number of WiFi frames in the environment. The number increases in dense urban areas where interference with other radio transmitters (both WiFi and non-WiFi) and bandwidth overlapping of multiple nearby networks cause frequent retransmissions of the frames. Notably, a considerable portion of WiFi frames are not data frames but control and management frames to control the physical medium and manage the basic service set (BSS) respectively. In this paper, we specifically focus on beacon frames, which are used for advertising access points and their capabilities.

Beacon frames are generated by access points so that that they can be discovered by other stations (STAs)[1]. Typically, beacons contain a Service Set Identifier (SSID), though this can be omitted to create "hidden" networks. 802.11 standard specifies that APs should emit multiple beacons per second so that they can be quickly discovered by stations passively scanning across all channels. Given the number of active access points in urban areas, the abundance of beacon frames alone creates dense traffic in all WiFi channels. In Figure 1, we have shown the average number of beacon frames per second in multiple locations of two cities in the US.

Covert channels are unconventional forms of transmitting data that are typically difficult to detect and interpret [40], [4], [9], [12], [18], [23], [31]. Existing covert channels in wireless technologies exploit either the physical layer characteristics of the medium and the radio packets or employ traditional ways of analyzing inter-packet-delays to exfiltrate covert data [8]. Multiple mitigation techniques have been proposed in literature to counter physical-level radio covert channels [29]. Both the implementation and mitigation of these techniques require special equipment.

In this work, we present a novel strategy, CHAOS, to implement a WiFi-based covert channel that uses beacon frames to encode and broadcast data. We do not modify the operations of the physical layer, nor do we change the regular

[1]Any client that communicates through IEEE 802.11 is a station.

Network and Distributed System Security (NDSS) Symposium 2025
24–28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230187
www.ndss-symposium.org

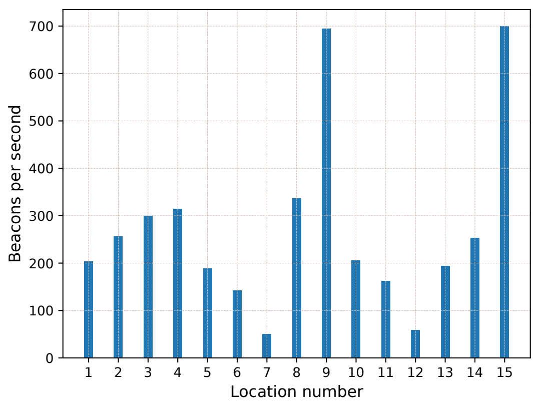


Fig. 1. Number of beacons in 15 distinct places in two US cities. Locations range from less crowded to dense areas.

timings of WiFi frames. CHAOS targets subtle timings of a *noise* component between beacon frames.

Our model establishes a reliable covert channel at the software level without violating the structural properties of the frames. We take advantage of the ambient noise that exists in all WiFi networks, and use the natural delays between management frames to encode and extract covert bits. CHAOS exploits the Timing Synchronization Function (TSF) of the individual STAs in beacon frames to encode timing data. TSF is the backbone of 802.11 time synchronization and controls many important tasks like signal scheduling and power management. In other words, WiFi will not function without TSF.

CHAOS broadcasts beacons for multiple virtual access points. It uses natural microsecond-precision jitters in TSF, which we refer to as Delay Levels (DLs), and are present between normal intervals of beacon frames. The order of the beacons coupled with the inter-frame noise that we deliberately introduce between frames creates a set of permutations that are mapped to covert bits by receivers. Beacon frames used by CHAOS comprise only a small part of all management frames (including other beacons) that exist in the air. CHAOS is a broadcast protocol; a secret message can be read by all in-range stations that are aware of the covert channel. The rest of the stations will see the frames as normal benign beacons. CHAOS is resistant to ambient noise. The timing patterns that we use in our covert channel look like normal inter-frame delay (IFD) timing drifts. Using CHAOS we can reliably broadcast covert payloads to multiple stations without causing any suspicion that any data transmission is taking place. From the perspective of an adversary, nothing is observed other than normal advertisement frames from already-ubiquitous access points.

CHAOS makes use of the inclusion of high-precision timestamps in WiFi management frames to create a timing channel that is both precise and easily managable by the receiving stations that are part of the covert communication.

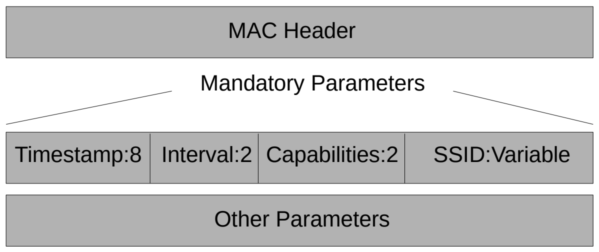


Fig. 2. Beacon frame format. Sizes are listed in bytes.

We also identified a previously-unknown opportunity to mount a correlation attack through careful observation of TSF noise: the jitter in TSF values from benign access points is affected in predictable ways by the load on that access point.

In short, we make the following contributions:

- We identify a new way of exploiting two important side effects of TSF counters: variable microsecond-precision noise and the effect of load on this noise,
- We introduce a method for producing timer jitter that follows statistical distributions seen in natural ambient TSF noise,
- We show an effective strategy for taking advantage of TSF noise while being resistant to noise at the same time,
- We implement and demonstrate a reliable covert channel to broadcast hidden payloads in a WiFi environment, and
- We describe a novel correlation attack using analysis of TSF noise patterns.

II. Background and Overview

In this section, we briefly explain the design and properties of TSF, protocol assumptions and requirements, and the consequences that can lead to exploitation of TSF.

A. *Time Synchronization in the 802.11 MAC Layer*

Access points advertise their presence to surrounding stations using beacons sent at constant intervals. The interval is usually set to a default universal value in all commercial access points. The 802.11 MAC layer standard (Part 11) [17] defines a time unit (TU) as 1024 microseconds. The interval between two consecutive beacons is called Target Beacon Transmission Time (TBTT). In most access points, TBTT is by default set to 100TUs which is 102,400 microseconds. 100TUs has been selected as a reasonable interval because longer durations can increase connection delays for stations that are waiting for a beacon and shorter durations will create extra frames and unnecessary burden on the network.

Figure 2 shows the abstract structure of the frame. Generally, it is assumed that an access point transmits a beacon frame *approximately* every 100 milliseconds. In this paper, we specifically look at the micro-second TSF noise that is reflected in the timestamp field the moment the TSF value is written in a beacon frame.

The 16 bit *Interval* field that is filled by the access point specifies the nominal delay between two consecutive beacons.

Crucially, the actual transmission interval is not necessarily equal to the defined interval because the shared medium must be free before a beacon can be transmitted. An access point checks the current state of the medium using CSMA/CA protocol. If the access point senses that the medium is busy it sets a random back-off timer and then tries again. The 64-bit *Timestamp* field of a beacon and probe response frame shows the access point's clock since it started (was turned on by the user). The MAC layer standard[17] states that *"In an infrastructure BSS or in a PBSS*[2]*, the AP in the infrastructure BSS or the PCP*[3] *in the PBSS shall be the timing master for the TSF."* The access point achieves the required timing goal through a beacon transmission operation which is described as *"In a non-DMG*[4] *and non-S1G BSS, the AP shall periodically transmit frames called Beacon frames."* Similarly, for 802.11ad we have *"In a DMG infrastructure BSS, zero or more DMG Beacon frames shall be generated for transmission by the AP every dot11BeaconPeriod TUs."*

The timestamp field of a beacon frame is populated by the value of the TSF timer. It is also stated in the standard that *"Each STA shall maintain a TSF timer with modulus* 2^{64} *counting in increments of microseconds."* The important thing about this value is that the TSF counter stored in the timestamp field is always prone to noise. Technically, the timestamp in a beacon or probe response frame is set to the value of the TSF clock, the moment that *"the first bit of the frame appears at the transmit antenna connector"*. This means that the timestamp will change in each transmission, not only due to waiting for a period of 100 TUs, but also because of the time that is lost due to processing, hardware environmental noise and waiting for the medium before the frame can be transmitted through antenna sectors. Another reason is thread suspension is usually not precise at microsecond level. For example, standard suspension functions like `sleep()` and `usleep()` guarantee that the calling thread is suspended for *at least* the requested time. The actual receipt times that are seen by the receivers always deviate significantly in microsecond or even millisecond granularity due to radio interference, hardware imperfections, distance and processing times at different levels during the course of transmitting and receiving the signals. All of these factors translate into random fluctuations in TSF-based timestamps.

On the other hand, this level of natural noise during synchronization is actually expected: The standard does not require that the inter-frame delays (IFDs) of beacons be precise. The standard does not assume that the stations receive beacons exactly at TBTT intervals and it does not mandate that the difference between two consecutive timestamp values must be precisely equal to "dot11BeaconPeriod TUs". This level of precision is not needed to implement the functionality that is defined for beacons. It is just required that all stations in the BSS be aware of the *current* TSF value of the AP at the time of beacon transmission for the purpose of synchronization. The reason that AP waits for a specific time before transmitting another beacon is solely limiting the number of beacons to a minimum so that other stations receive sufficient beacons for scanning and time synchronization. As long as at least a handful of beacons per second advertise the existence of an access point along with the access point's TSF timer, other STAs can find the access point in a normal passive scan and perform TSF synchronization. The existing TSF noise does not cause any problem in this process but it does create side effects that can be exploited.

[2]A BSS is the short form of "Basic Service Set" which is a regular WiFi network and PBSS means Personal BSS like an ad-hoc network.

[3]PBSS Control Point.

[4]DMG: Directional Multi Gigabit

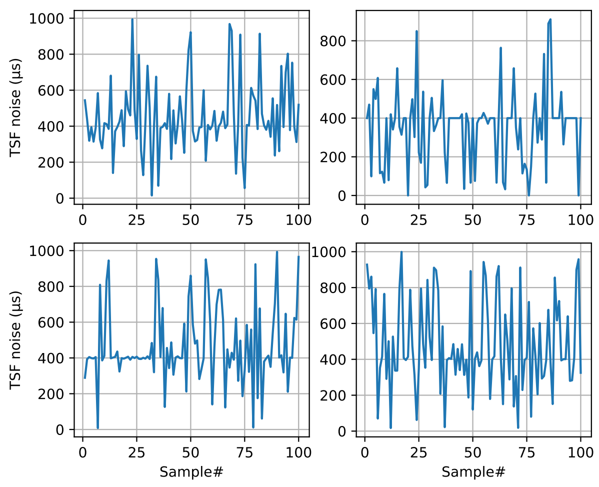


Fig. 3. TSF noise sampled 100 times for 4 different access points.

The way TSF synchronization has been designed created—inadvertently—an opportunity for us to build a reliable and relatively fast timing-based covert channel which is extremely hard to detect or stop.

B. *TSF Noise and Noise Resistance*

The natural noise that is observed in timestamps of beacon frames is the basis of developing CHAOS. Since the foundation of CHAOS is built upon ambient noise, our approach is noise-resistant. We mimic the TSF noise that normally exists. Increasing the noise does not affect the functionality of CHAOS because it adapts to the new noise observed in the environment.

In Figure 3 we have shown the TSF noise from four access points made by different manufacturers. Basically all access points in all channels behave quite similarly in terms of showing this specific type of noise. We measure TSF noise at the granularity of microseconds, and its value varies significantly in different frames. However, it is usually bound within the range of a few to about 1,000 microseconds.

Because the TSF counter is included directly in beacon headers, placing and recovering this noise does not require special hardware capable of high-precision time measurement: any off-the-shelf WiFi chipset that provides raw access to beacon frames (which is many of chips) can be used.

C. Hiding in the Crowd

Another thing that assists CHAOS to evade detection is the high volume of benign frames in the air. The 2.4GHz band is broken up to 14 channels, each of which is 25MHz wide. Among them, three channels in the 2.4GHz band are non-overlapping; 1, 6, 11. The majority of access points on this band, choose one of these three channels by default. In the 5GHz band there are 24 non-overlapping 20MHz channels. Like the 2.4GHz band, in most countries only a couple of all the available channels are used for majority of WiFi communication. In urban areas, collectively there are tens of thousands of beacon frames per second in both bands. This number increases significantly in crowded environments like a city downtown or densely populated locations within an urban area. CHAOS only uses a fraction of the existing advertisement frames to transmit coded secret messages.

III. Design

CHAOS comprises one broadcast transmitter and multiple receivers. CHAOS creates a list, β, of m MAC addresses which represent m access points to advertise ($\beta = \{AP_1, AP_2, ..., AP_m\}$) using normal beacon frames the way regular access points are advertised. These are the access points that are used to establish our covert communication. We do not need a separate physical access point for each access point defined in the system. All beacons are managed by and transmitted from *the same* machine. In other words, one physical transmitter is enough to run CHAOS. Note that, to change the physical characteristics of the beacon frames across different CHAOS APs, we can use a separate *network card* for each AP or for a group of APs. For example, AP_1 and AP_2 can be assigned to two different USB NICs, $iface_1$ and $iface_2$, each of which employing a different WiFi chip. While the USB cards can be placed in two different locations, they will be connected to the same machine and governed by the same instance of CHAOS.

The receiver and transmitter have the same β set. Then a subset of $n \leq m$ access points, A, are chosen from β so that $A = \{ap_1, ap_2, ..., ap_n\}$ where ap_i can be any of the access points defined in β. These n access points defined in A are used for secret communication in each *burst*. CHAOS can dynamically change members in A at each burst, rotating the access points to be placed in this subset.

burst: A burst is the transmission of m beacons to advertise m APs defined in β.

We call an access point that is a member of A, a TXAP (transmission access point), and an access point that is in β and not in A is called a CAP (cover access point). Secret bits are transmitted by TXAPs. CAPs are used to increase the cost of statistical analysis of timing patterns to make it more difficult for an adversary to locate a CHAOS transmitter. For the most of the rest of this section, we use *AP* and *STA* for an access point and a station respectively to save space.

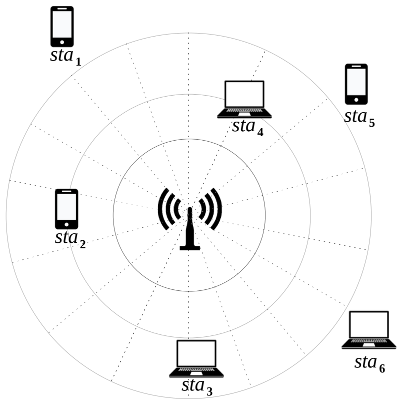


Fig. 4. TXAPs encode the secret message in the form of two permutation components that are sent using the beacon frames. All STAs that are within range can capture the frames (sta_2, sta_3 and sta_4). CHAOS STAs will decode the components and recover the secret message.

For each AP, we advertise its MAC address and capabilities using standard beacons. Following the normal functionality of regular APs, we send a beacon frame for an AP every 100TUs or 102ms. We call each transmission cycle a burst.

In each burst, the transmitter and the receiver have the option to change the APs defined in A by choosing a different subset from β using a key known to both sides. The transmitter and the receiver will always have the same members in A and β at any given burst. The TSF noise of a CAP is not used for covert transmission and its value is selected from a normal distribution as we explain later. For TXAPs, however, we follow a specific strategy to create the TSF noise.

In each burst, we define n slots specifically for the n APs in A that are used to carry secret bits in the current burst. Each slot is assigned to one AP and we transmit the beacons starting from the first (s_1) to the last slot (s_n). The assignment of the APs changes in each burst, and the order of the assignments in each burst creates a unique sequence which constitutes the first component of permutations that are mapped to secret bits. The TXAP frames can be sent in the same 802.11 radio channel or across multiple channels and as mentioned before, using the same or different interfaces. Figure 4 shows a simple model of CHAOS stations receiving the coded timing noise in a WiFi network.

For each slot, we define a *delay level* which mimics the natural ambient TSF noise. Assume that ts_i and ts_{i+1} are the timestamps of two consecutive beacons from a specific AP. Normally the following equation relates these two values in an 802.11 access point:

$$ts_{i+1} = ts_i + 100TU + \varepsilon \qquad (1)$$

Where ε is the variable TSF noise that usually ranges from a few microseconds to one millisecond. This is what we refer to as ***TSF noise*** in this paper and this is the noise using which we transmit covert data.

TSF Noise: The variable microsecond-level deviation from TBTT in beacon intervals measured using the timestamps of two consecutive beacon frames.

The delay level (dl) in CHAOS is mapped to the ε component of the equation and in each burst it is defined as:

$$dl = l \times DS + \varepsilon_i \tag{2}$$

Where DS is called *Delay Step* which is a constant value and ε_i is a random value in the range $[0, DL/2]$. The most important part of this equation is l which constitutes the second component of our permutation mapping. The receiver reverses the above operation to find l. This coefficient is an integer that is chosen from the range of $[0, DL\_NMAX)$ in which DL_NMAX is a constant that specifies the total number of *Delay Steps* that l can be chosen from. We use the short form of capital L to point to DL_NMAX. This means that l is a number from $\{0, 1, .., L-1\}$ set. The magnitude of L directly specifies the size of the permutation space. The values of DS and L are chosen and defined for the system in a way that the following inequality is satisfied:

$$DS \times L < TSF\_MAX \tag{3}$$

Where TSF_MAX is the average maximum TSF noise that we have observed in various APs. Basically, DL and L have an inverse relation; if we choose a bigger value for L we have to reduce DS.

After each dl is calculated, it is added to the base standard TBTT (called *BASE DELAY* in CHAOS) and then is written in the timestamp of the beacon of the current slot. The sequence of the resulting dls will look exactly like natural TSF noise. A normal station will perceive these beacons as normal advertisement frames but the CHAOS receivers will recover the set of dls and combine them with the order of APs in the current burst to decode the secret payload.

A. The Permutation Space

The combination of the *order of the TXAPs* and the *TSF noise of each TXAP* in each *burst* creates a unique sequence that constitutes one permutation in our permutation space. For n TXAPs in A we can make $n!$ permutations and for L possible values for each TSF noise, we will have L^n permutations. This gives us $T = n! \times L^n$ total permutations. Therefore, in each burst, we can transmit U bits where:

$$U = \log_2 T = \log_2 (n! \times L^n) \tag{4}$$

Numeric representations of these permutations are indexed from 0 to $T-1$ to map each permutation to a unique sequence of secret bits.

We have shown a simple example in Figure 5 where $n = 5$. There are three bursts in this figure. In the first burst we have the sequence of $ap_2, ap_1, ap_4, ap_3, ap_5$ which is represented as sequence $(2, 1, 4, 3, 5)$. Only CHAOS stations know the correct assignment of the order of access points to permutation sequences as we define for the system what access point should

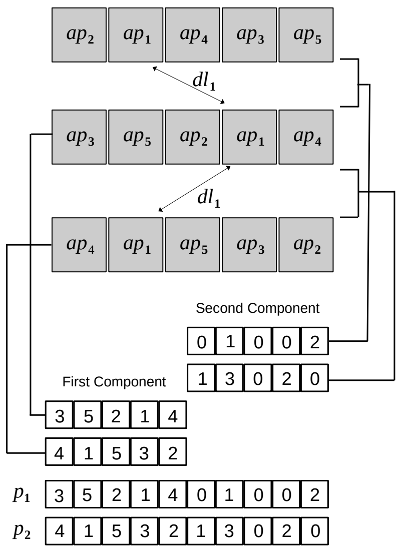

ap_2 ap_1 ap_4 ap_3 ap_5
dl_1
ap_3 ap_5 ap_2 ap_1 ap_4
dl_1
ap_4 ap_1 ap_5 ap_3 ap_2
Second Component
0 1 0 0 2
1 3 0 2 0
First Component
3 5 2 1 4
4 1 5 3 2
p_1 3 5 2 1 4 0 1 0 0 2
p_2 4 1 5 3 2 1 3 0 2 0

Fig. 5. TXAPs are assigned to transmission slots and TSF noise is defined for each slot independently. Delay levels and TXAP order create permutations (p_1, p_2) that are mapped to secret bits. The burst is broadcasted and all CHAOS receivers that see the frames recover the permutation components to remap the burst to the covert payload. Each delay level is calculated between two beacons of the same AP (For example, dl_1 for ap_1 in the figure.). TXAP order assignment is only known to CHAOS stations.

be mapped to what number. For example, if MAC address m_i represents AP_j, only CHAOS stations can place AP_j in the corresponding sequence when the source address of an RX beacon is set to m_i.

In the second and third bursts we have $(3, 5, 2, 1, 4)$ and $(4, 1, 5, 3, 2)$ respectively. The value of the dl component is calculated relatively between two frames. For the transition between the first and second burst we have a delay level sequence of $(0, 1, 0, 0, 2)$ and for the next one we have $(1, 3, 0, 2, 0)$. This will yield two permutations of $(3, 5, 2, 1, 4, 0, 1, 0, 0, 2)$ and $(4, 1, 5, 3, 2, 1, 3, 0, 2, 0)$ for the receivers. These are broadcast covert messages. The receivers do not need to associate with the APs or even transmit any frame on any channel. Consequently, even the transmitter does not know where the receivers are located or which stations can read the messages. This means that even if an adversary can detect a transmitter, they will not be able to identify the receivers. We will present a comprehensive analysis of detection possibilities of CHAOS APs in Section IV.

In Section III-D, we have included a step-by-step example of the encoding strategy to convert data bits to CHAOS permutation components.

B. The choice of CHAOS parameters; m and n

Selecting different values for m, n, DS and L affects the covert communication speed and how difficult the detection of the system will be. Choosing DS and L is relatively straightforward. Bigger values of L increase the covert bandwidth. Mathematically, this also reduces the range of values that we can use for DS. But apart from speed, it does not affect the covert channel.

The selection of m and n, however, is significantly different and requires careful analysis. n is the main number to change U in Equation 4 and hence the main factor in determining speed. That said, choosing a too big value for n increases burst miss rate (explained later) and we need a limit on that. m on the other hand, specifies the statistical properties of the TSF noise that is generated by CHAOS. We want this noise to look similar to the noise that we see from other (benign) access points. The reason that m specifies the distribution of the TSF noise, is that TXAPs will periodically change and the noise that is seen for a specific access point does not purely depend on the distribution of bits in the secret message. Remember that as we mentioned at the beginning of this section, the transmitter and the receiver have the option to rotate the APs in A using a shared key.

We are not limited to any specific strategy of choosing the members of A. The transmitter and the receiver can use any selection algorithm as long as they end up with having the same members in A in each burst. Note that the burst number in the transmitter and all receivers is always the same and synchronized through observing the difference between two consecutive timestamps of any AP used in CHAOS. For example, at the receiver side, if $ts_{i+1} - ts_i \geq 2TBTT$ it means that the transmitter is one burst ahead of the receiver and the receiver increments its current burst number by one and so on. To help the reader better understand the AP rotation process, here we present a simple selection strategy to choose APs for A in each burst.

A simple TXAP rotation example

In this example we assume $m = 20$ and $n = 5$. Access points are numbered from 0 to 19 in β (or B in code snippet 1). When the code starts we use the shared SECRET to seed the number generator which is used to shuffle B using Fisher Yates algorithm. Whenever we increment the burst number we shuffle B and take the first n integers out to fill A. The code snippet of this rotator is written below.

If we set SECRET to an arbitrary number like `0xA0` (decimal 160), Listing 2 shows the first five sequences of A that the transmitter and receiver will generate.

```
void init(int SECRET){
  srand(SECRET);
}
int * build_A(int burst_increment_steps){
  int i;
  u8 *A=malloc(n);
  for (i=0;i<burst_increment_steps;i++)
    shuffle_fisher_yates(B);
  for (i=0;i<n;i++){
    append(A,B[i]);
  }
  return A;
}
```

Listing 1. Example code snippet of a simple TXAP rotator

These numbers that fill A specify the APs for the first five bursts. For example, in the second burst, the transmitter will use $\{ap_0, ap_5, ap_7, ap_8, ap_{13}\}$ as the set of TXAPs, and the receiver will use the TSF noise of the same APs to decode the permutation components.

```
5 13 16 4 15
0 5 8 7 13
11 16 10 0 12
7 2 8 4 10
18 1 7 8 12
```

Listing 2. The first 5 sequences generated with the given SECRET as integer 5.

C. The Problem of Missed Bursts

In radio communication, it is common for a receiver to miss some of the transmitter's signals. WiFi networks are no exception. While frame miss rate, is different for different chips and drivers, all network cards miss some frames. In CHAOS, since the communication happens in a one-way broadcast manner, the receivers cannot acknowledge frames. An entire burst is missed, if at least one of the beacons of the current TXAPs is not received at the receiver side.

We resolve this problem by repeating the transmission over the covert data *one cycle at a time* with a known fixed size instead of sending the payload only once. In each burst, we send one unit of data which is equal to T bits, described in Equation 4. Effectively the covert data is divided into sequential units and the transmitter sends them in the order they are in the payload. The size of the covert payload must be known to both sides, and consequently the receiver and the transmitter know how many units should be transferred for the payload. Since CHAOS transmitter and receivers are synchronized in terms of burst numbers, whenever a receiver detects that a burst was missed, it moves to the next burst and tries to fill the next data unit in the RX buffer. When the transmitter hits the end of the payload buffer, it starts retransmitting the payload from scratch by moving to the first data unit and the receivers will fill the missed units in the next cycles.

For example, if $T = 16$ and the covert payload is a 24-byte secret message, we need 12 bursts to send the whole message. Assume that, the receiver misses the third burst. This will be detected immediately after the receiver captures a beacon from one of the CHAOS APs after the third burst is completed at the transmitter side. The receiver then proceeds to fill the next parts of the secret message while the transmitter is sending the remaining 9 units. Eventually after the transmitter gets back to $burst_3$ in the next cycle, the missed data unit will be retried by the receiver. By repeating this process, it is guaranteed that all missed bursts are recovered by all receiving STAs.

D. Encoding Example

We use a simple example in this section to show the encoding strategy for converting secret bits to TSF noise and vice versa. In this example, we assume there are six access points (ap_0 to ap_5) in β and only three rotating access points in A ($n = 3$ and $m = 6$). Furthermore, we assume there are only nine delay levels for each access point ($L = 9$). We also assume that the sample secret bytes that we want to

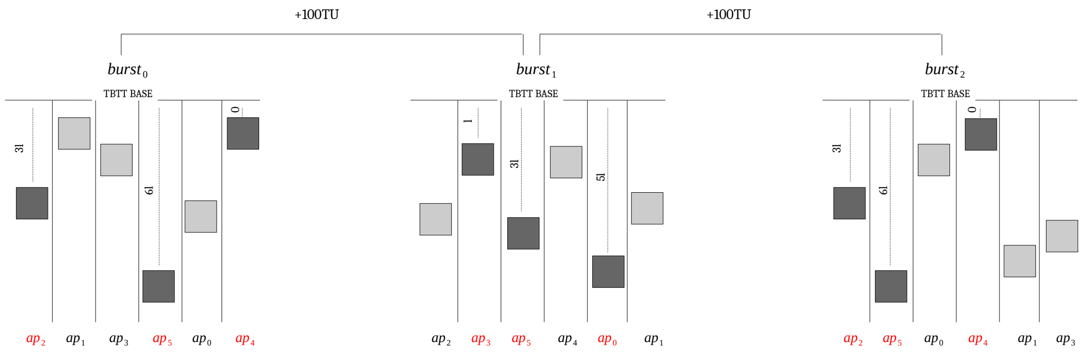
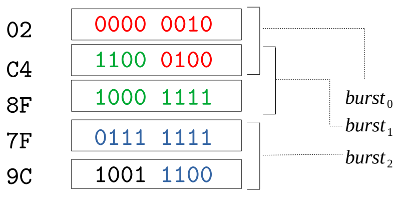

+100TU
+100TU
burst_0
burst_1
burst_2
TBTT BASE
TBTT BASE
TBTT BASE
3l
6l
0
1
3l
5l
3l
6l
0
ap_2 ap_1 ap_3 ap_5 ap_0 ap_4
ap_2 ap_3 ap_5 ap_4 ap_0 ap_1
ap_2 ap_5 ap_0 ap_4 ap_1 ap_3

Fig. 6. Arranging the beacons for the first three bursts to carry the sample payload. CAP APs can take arbitrary orders in each bursts.



TX burst	burst payload	components
0	`0100 0000 0010`	`1,(360)`
1	`1000 1111 1100`	`3,(135)`
2	`1100 0111 1111`	`4,(344)`

Fig. 7. Breakdown of secret payload and grouping the bits to map permutation components.

carry using CHAOS is `02 C4 8F 7F 9C` with 02 being the lowest address byte of the payload in memory.

For three access points and nine delay levels we will have a sample space of $6 * (9)^3 = 4374$ members. This number is enough to transmit 12 bits per burst. The number is broken down into two sample spaces for two permutation components: 6 for the first component and $9^3 = 729$ for the second component.

The secret payload is 40 bits, and in this example we discuss the first three bursts which will map the right-most 36 bits (starting with $0x02$). In Figure 7 we have shown how the first three groups of bits are broken down to map to the first three bursts.

Access points in A are assigned to three slots s_0, s_1 and s_2. The order of the access points is interpreted based on the order of the transmitted slots. The specific order-to-permutation assignment strategy is defined arbitrarily and is part of the secret shared between stations. In this example, we define the first permutation of the possible orders as the order in which the access points are assigned to the slots in ascending indices from left to right, and then the next permutations will follow the same pattern similar to generating an ascending sequence of continuous natural numbers. For example, in $burst_n$ if $A = \{ap_2, ap_3, ap_5\}$, then the first permutation is when $s_0 = ap_2$, $s_1 = ap_3$ and $s_2 = ap_5$. If these three access points are transmitted in another order, say $ap_2 ap_5 ap_3$ the receiver will interpret it as the second permutation of the *order* component. The order of CAP APs can be arbitrarily set in each burst.

The first twelve bits are `0100 0000 0010` which is equal to decimal 1026. To get the first permutation component we divide 1026 by the size of the second component and get $1026/729 = 1$. This means that we will create the permutation of the slots in a way that the first component in this burst will map to *second* member in its sample space and the second component which is the TSF noise will map to the element $1026\%729 + 1 = 298$ in the second component. Since the assignment of access pints in A depends on the secret key, here we assume an arbitrary assignment for the first three bursts that is shown in listing 3:

```
2 4 5
0 3 5
1 2 4
```

Listing 3. The first three TX rotation orders.

For the first burst, $A = ap_2, ap_4, ap_5$, and since we want the *second* member of the order component, we set the order of the access points as $\{2, 5, 4\}$. For the second component which is the TSF delay levels, 297 modulus 9 is 360. So we will create the sequence of $\{3l, 6l, 0\}$ respectively for $A = ap_2, ap_4, ap_5$. This will result in the following TX sequence:

$$burst_0 = \{ap_2 : 3l, ap_5 : 6l, ap_4 : 0\}$$

.

More precisely, while it is not shown in Figure 6, a random value is added to each resulting delay level so that the created noise is not an integer multiple of l (refer to Equation). At each new burst, TSF timestamp is incremented by 100TU (theoretical part) plus the delay level (noise part). We assume the theoretical base timestamp for the first burst is T_0. We trace discrete increments of $timestamp/100TUs$ to track bursts. For example, we will have $T_0 + 100TU$ as the base timestamp for the second burst.

The delay levels of ap_0, ap_1 and ap_3 at the first burst are chosen from a random variable with normal distribution. Remember l is chosen based on Equation 3.

Similarly we will have $\{ap_3 : l, ap_5 : 3l, ap_0 : 5l\}$ and $\{ap_4 : 3l, ap_1 : 4l, ap_2 : 4l\}$ for the second and third bursts respectively as shown in Figure 6. The receiver will reverse these steps and recover the exact same sequences to form the first three 12-bit groups of the payload. In the case of missing a beacon frame, the corresponding burst is marked an incomplete. The receiver will mark a burst as incomplete, if the base timestamp exceeds the expected range of the current burst. For example, if the receiver is waiting for the second burst and it receives a beacon with a TSF timestamp in the range of $T_0 + 200TU$ and $T_0 + 300TU$, it moves to next buffer cell which corresponds to the third and not second burst. The second burst will be recovered at the next cycle of TX rotation.

E. The Threat Model

A receiver and a transmitter:

- have the same values for the permutation parameters: m, n, L, BD, DS, and
- have the same constant members in β and the same variable members in A in each burst, and
- do not have mutual knowledge about each other.

Note that, the receivers capture the beacons that are used for covert transmission but they do not know what station is sending them[5]. Similarly, the transmitter does not have any knowledge about who the receivers are. The permutation parameters, shared key and β are not transmitted at any time. They are defined for the system as a set of pre-selected parameters. These parameters can arbitrary change between different compiles if we need different groups of stations use separate keys or configurations to establish independent communication channels in those groups.

The adversary:

- can listen to and capture all WiFi frames in all WiFi channels[6], and
- does not know whether CHAOS is being used, and
- does not posses any knowledge about the location and addresses of CHAOS STAs, and
- does not possess any knowledge about β, A and the shared key between CHAOS STAs, and
- since it does not know the shared key, it will not be able to reproduce the TXAP rotation sequences.

An adversary might try to achieve any of these goals:

- detection of whether CHAOS is being used,
- upon detection (or prior knowledge about usage of CHAOS in the environment), trying to recover covert data,
- stopping CHAOS or interrupting the covert communicatoin.

We discuss all these possibilities in the next sections.

[5]The raw frames used by CHAOS do not contain any identifying information of the transmitter.

[6]Usually, this is not possible due to environment noise and hardware imperfection. But we assume a strong adversary with an ideal capture system and zero percent frame miss rate.

F. Some notes on the choices of AP details

Normally, 802.11 access points have different MAC addresses and SSIDs. A simple scan on any WiFi channel finds many access points and a significant number of them are hidden. An access point is hidden if the beacons that it transmits do not have an SSID. More precisely the length of the SSID in the advertisement frame is zero. Users that want to connect to a hidden station must provide the correct SSID before they can complete the association stage. A WiFi router can create multiple access points on 2.4GHz and 5Ghz bands. Some of hidden access points we find in our scans, have similar MAC addresses to a visible access point with only the 4 right-most bits being different. This suggests that the related beacons are transmitted by the same device. Irrespective of what the sources of hidden APs are, this type of access points are quite prevalent.

CHAOS creates a MAC address for each access point that it advertises. Similarly we need to decide what SSID should be used for each access point. A reasonable strategy is to make some of the APs hidden and some of them visible. This has two benefits. First, we will have a combination of both types of SSIDs (zero and non-zero lengths) like benign access points in the environment, and second, we will need to come up with fewer number of SSIDs to be used for advertising the visible APs. Another strategy, is to remove SSID from all CHAOS beacon frames which will effectively make them hidden access points. Given the high number of hidden access points in urban areas, this is a valid solution for small values of n. For those APs that we want to have a valid non-zero SSID, we choose SSIDs the same way people choose a WiFi name for their home or business network. Note that SSIDs and whether our APs are hidden or visible, do not have anything to do with TSF and noise analysis and the functionality of CHAOS. SSIDs only specify how APs show up when others scan surrounding WiFi stations.

IV. Evaluation

In this section we evaluate the system in terms of two main aspects: Transmission speed and Detectability. In Sections IV-C and V we will discuss some possible ways to mitigate this type of covert channel. We have implemented CHAOS under Linux. We used standard consumer on-board and USB WiFi NICs as the physical interfaces for transmission and receipt of WiFi frames. The software layer is implemented directly at user-space through Linux raw sockets which manage the WiFi interfaces on monitor mode.

A. Covert Transmission and CHAOS Data Rate

The two parameters that affect speed are n and L. L increases the choices for each delay level. n has a more significant overall effect mathematically. We have visualized in Figure 8, the effect of both of these parameters on the nominal speed that we get using Equation 4. However, there are some important limitations in choosing both n and L.

While n has a bigger impact in Equation 4, practically, increasing n will also increase the average number of TXAP

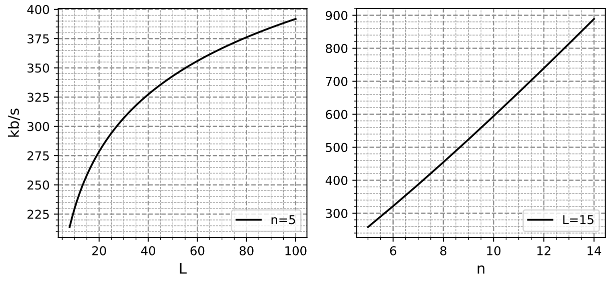


Fig. 8. Data rate of the covert channel in two examples when we fix one of the parameters and change the other one. We have set $n = 5$ for the first graph on the left and $L = 15$ for the other one.

missed frames per burst and consequently the number of retries needed to fill a missed data unit. The reason is that the integrity of a data unit in any burst depends on all of the TXAP beacons in that burst. If we increase the number of these beacons in each burst, there will be a higher chance that at least one of the beacons is not captured by a receiver. Furthermore, we want our beacon frames to be a small part of existing beacons in the air. So we need to keep n relatively small. The other problem with increasing n, is overflowing the permutation space which causes various implementation issues. A similar issue also exists with L but to a lesser degree. The important thing about L is that increasing L implies smaller values for DS. On the other hand, we want each delay level to be at least a few microseconds. Anything less than one microsecond is not acceptable. Overall, the transmission speed vary largely based on the values of these parameters.

We have observed the result of many choices for (n, L) tuple in hundreds of tests. Setting n and L to 6 and 216 respectively will create a good balance between speed, undetectability and cost, and yields acceptable results. With these values we have around 73 trillion permutations which is enough to transmit 56 bits in each burst. The average data rate that we achieve using this setup is 520 b/s which is significant for a timing covert channel. This speed is enough to transmit a 2048-bit RSA private key out of an air-gapped local network using a Raspberry Pie within 26 seconds. The channel stays stable in environments with high radio noise and interference.

The reason we use average rates is that data rate normally changes in different tests mainly because of radio interference in the environment which affects the number of missed frames at the receiver side. Also note that this is the bitrate that we get when we use only 6 beacons for data transmission in each burst.

The isolated TSF noise of a random TXAP, is shown in Figure 10 when we transmit 4 different covert payloads. *Isolated noise* or *data-dependent noise* is the noise that we see when we do not change the set "A" between bursts. Therefore, all the noise values will be solely affected by the distribution of bits in the secret data that we send. In other words, this noise is observed when $m = n$. If $m > n$, we call the TSF noise of a CHAOS STA, *randomized* or *normalized* noise. In the following, we explain the effects of these noise types on the

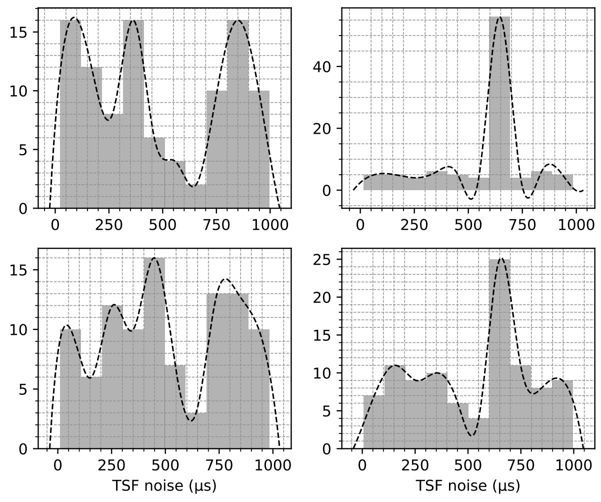


Fig. 9. Histograms of TSF noise of transmitting 4 random files with 4 different file formats using CHAOS. 100 samples with cubic curve visualization. Formats were *jpg, zip, elf, gif*.

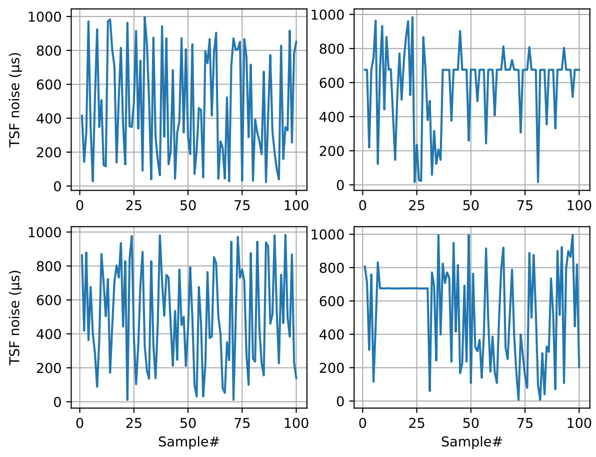


Fig. 10. Raw TSF noise of the same samples shown in Figure 9.

detectability of CHAOS. Note that in all figures in this section wherever we report noise that was generated by CHAOS, the noise belongs to a random AP $ap_x \in \beta$. All APs in CHAOS can be equally used for noise assessment and we target one of them randomly to analyze the noise patterns. An adversary will have to do the same thing with all access points in the area.

B. Detection

In an urban area with thousands of WiFi stations (including many access points) at any location, continuously transmitting frames, detecting the existence of any covert channel used by an unknown station is quite challenging in general. In the case of CHAOS, the challenges are even more significant because the secret bits are sent using permutation components derived from the order of a select set of benign advertisement frames and the pattern of a timestamp noise that already exists in all other advertisement frames.

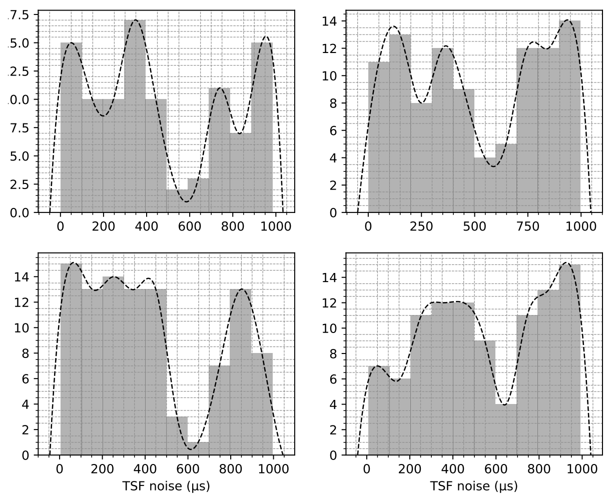


Fig. 11. TSF noise of the same files after adding a layer of encryption.

To detect the existence of CHAOS, an adversary will have to identify the access points that are advertised by CHAOS through analyzing the TSF noise of all access points in the environment and then try to distinguish regular TSF noise versus TSF noise used in CHAOS.

We discuss the topic of detection of the system in three parts:

- The effect of TXAP rotation and m/n ratio,
- the relation between channel data rate and detectability,
- adversarial AP fingerprinting at the physical layer.

1) TXAP rotation and noise distribution: Looking back at Figures 10 and 3, we can infer some visual and statistical numeric differences between the isolated noise of a CHAOS STA and the normal TSF noise from a regular access point. We want to minimize these differences down to a point that CHAOS STAs will be indistinguishable from regular access points. To ensure that the noise that is generated by CHAOS is similar to benign TSF noise in normal access points, we carefully studied noise distribution and numerical characteristics in various experiments. We present and discuss the results in this section.

In Figure 9 we have shown the histograms of the noise sets reported in Figure 10 to investigate the distribution of the generated noise. Note that TSF noise in CHAOS is created based on the covert payload *and* some random parameters, while real TSF noise in other access points is generated only due to natural factors that we discussed earlier. One way to change the distribution of bits is encrypting the payload. We have shown the generated noise for transmission of the same four files after we encrypt them using AES in Figure 11. However, encryption is not enough to foil adversarial statistical analysis and achieve the undetectability that we need.

In Figure 12, we have shown the histograms of TSF noise from 16 random normal access points. Interestingly, we can see that the natural TSF noise observed in regular access points, follows a distribution pattern in which most numbers are clustered around one or two random peak values. While we do not see a bell curve in these plots, our observations show that TSF noise shares some important properties of a normal distribution when the AP is not under a high load[7]. Most notably, TSF noise of a regular access point did not present a uniform (truly random) distribution in any of our tests. Regardless, we need to make the generated noise in CHAOS to follow similar patterns as natural TSF noise. Obviously, encryption cannot provide the distributions that we want. This is where our TXAP rotation technique, resolves the issue.

TXAP rotation limits the number of times data-dependent noise is used for an access point in β. In each burst, if an AP is not chosen to be placed in A set, the TSF noise of the AP in that burst is chosen from a normal random variable (a random variable with Gaussian distribution). We can alternatively use standard suspension functions like 'usleep()' and measure the amount of microsecond-level extra suspension when the function returns to choose a random value for TSF noise. The imprecision of these functions also have a normal distribution. The combination of dependent and independent noise that form the sequence of timestamp values for each CHAOS AP, will create patterns that match benign TSF noise.

The ratio of n vs m, manages the frequency of choosing an AP in A. This frequency on the other hand, affects the TSF noise distribution. Clearly, when we increase the size of β, the overall effect of data-dependent noise decreases because each CHAOS AP is chosen less frequently to convey data-dependent noise.

Here we investigate the effect of TXAP rotation in different setups of n/m. In Figure 13 we have shown TSF noise of a random AP in β, generated by CHAOS with multiple values for (n, m) touple for two of the sample payloads used in the previous test (to save space we did not add the similar results for the other two file types). The ratio of n/m is written under each plot.

The numeric representation of benign TSF noise shows a semi-normal distribution with average statistical values as written below in Table I. Therefore, we should aim to reach the same statistical representation in CHAOS when we carry out covert payload. Particularly, the standard deviation of the noise must decrease. In Table II, we showed how the statistical values shift when we gradually change the n/m ratio from 1 to 0.24.

The standard deviation of TSF noise on average decreases 38% and 57% when we change the ratio from 1 to 0.43 and 0.24 respectively.

The statistical comparison shows that for ratios smaller than 0.43 (numerically, about 40% reduction in the standard deviation) we get an excellent normalization and end up with noise values that are quite similar to natural noise of regular access points.

2) Data rate and detectability: The detection of the system depends on the *ratio* of n to m. We showed that the system achieves strong stealthiness with a a ratio of 0.4. Note that

[7]As we explain in Section A the distribution of TSF noise changes by increasing traffic

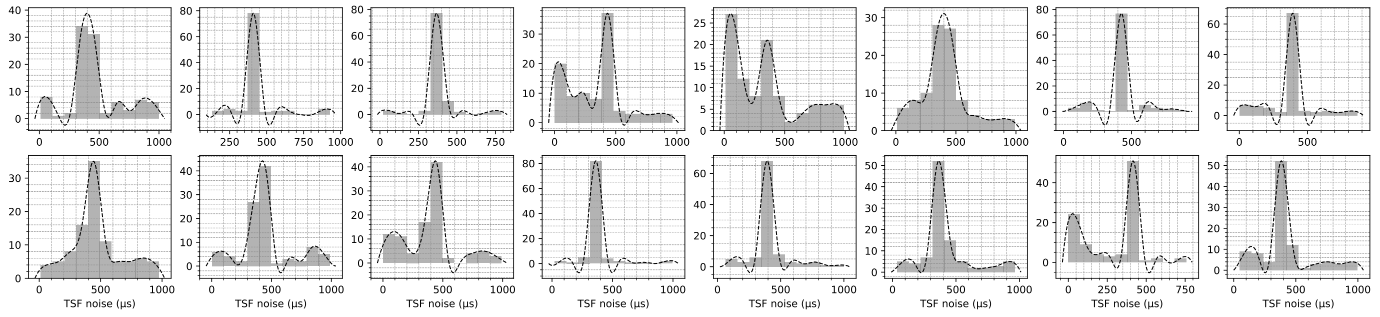


Fig. 12. Histograms of TSF noise of sixteen random access points from different manufacturers. Most values are clustered around a few points resembling a normal distribution.

TABLE I
NOISE AVERAGE AND STANDARD DEVIATION IN 10 RANDOM RUNS OF NORMAL TSF NOISE FROM MULTIPLE ACCESS POINTS.

Sample	Average	Standard deviation
1	445.46	215
2	468.29	217
3	419.99	123
4	399.97	99
5	334.12	215
6	362.29	278
7	409.35	194
8	405.56	130
9	381.32	157
10	454.95	205

TABLE II
THE EFFECT OF CHANGING n/m RATIO ON STATISTICAL PROPERTIES OF NOISE.

	$\mu = 500$		$\mu = 400$	
n/m	Average	Standard deviation	Average	Standard deviation
1.0	518.73	292	509.33	285
0.6	505.17	238	483.77	259
0.5	518.62	236	486.41	221
0.43	526.1	206	467.98	211
0.4	506.34	166	415.93	184
0.33	501.0	170	443.15	153
0.30	518.46	161	428.8	173
0.27	516.78	163	453.01	189
0.26	539.38	168	437.3	144
0.24	491.71	129	404.64	122

to adjust the ratio, we only changed the size of β not n. Consequently, the transmission speed would not be affected. As we explained before, the data rate is a function of n and L, and it is independent from m. This has two important implications:

- We can reduce the chances of detection of a CHAOS transmitter by adjusting m alone and leaving other parameters unchanged.
- Increasing speed by choosing a bigger value for n warrants a bigger value also for m to reach the same undetectability levels.

Therefore, the purpose of TXAP rotation is two folds: Spreading the covert bits over a variable list of access points to make the process of decoding the covert bits impossible without having the secret key even if an adversary knows all members of β, *and* making CHAOS-generated TSF noise similar to real TSF noise. Also note that, the adversary does not know the correct assignment of the orders of access points to the first permutation component. It means that in Figure 5, the adversary would not know, for example, which access point is considered ap_1 by CHAOS. Therefore, even a compromised TXAP key will not be enough for an adversary to be able to recover covert data.

In Figure 14 we have shown the generated noise (both raw and histograms) for transmission of a video frame from a surveillance camera with the $(n = 6, m = 20)$ setup. As we mentioned earlier, we get an average of 520b/s data rate with these parameters. Note that stealth communication is the main goal of this design, and we keep n and m small to make detection of TXAPs much more difficult. On the other hand, achieving significantly higher speeds is possible if we increase n to values greater than 10.

3) Physical layer fingerprinting: All properties of WiFi frames at the software layer are directly controlled by CHAOS. Hardware properties of frames, however, depend on the hardware infrastructure. Adversarial analysis of physical characteristics of beacons is one of the potential strategies to investigate the existence of a covert channel. Signal power and other low-level hardware-related signal properties can correlate a set of beacons to one transmitter. Some of these radiometric properties that have been studied in the past include frequency, amplitude and phase [26], RF fingerprints, sync correlation [5], [22], I/Q offset [5], magnitude and phase errors [22], power amplifier and frame interval distribution fingerprints [21]. Some physical properties can vary even between stations with the same model, made by the same manufacturer and running the same firmware [21]. If an adversary carefully investigates physical characteristics of the carrier signals, he can differentiate between beacons that have been transmitted from multiple stations even if they carry the same bits at the software layer. Ergo, an adversary would expect to be able to associate beacons of two different access points to two different physical transmitters.

We address this issue by implementing a flexible low-level hardware assignment using which different APs can be advertised by separate hardware. Radio frames in CHAOS can

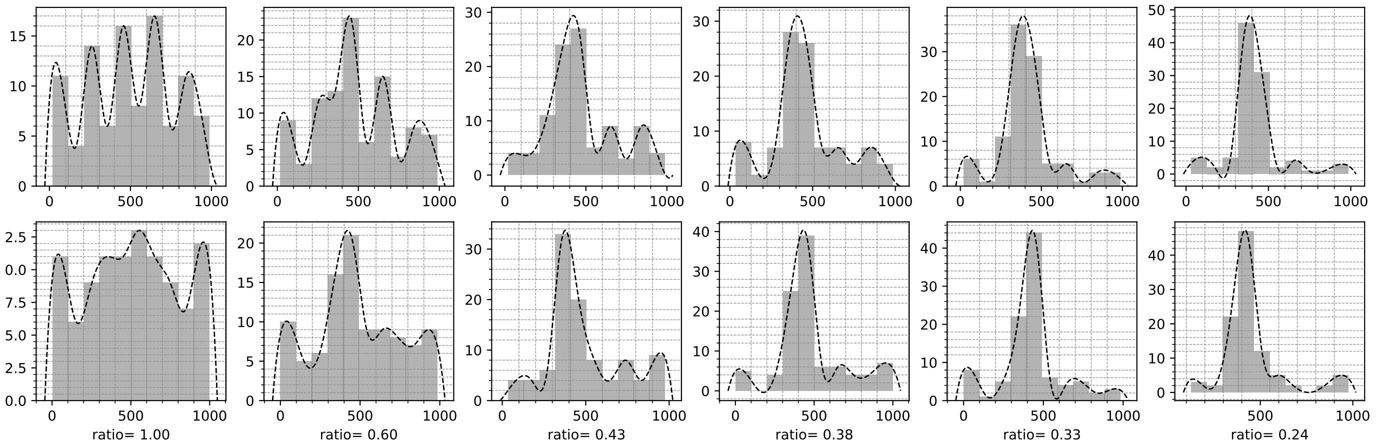


Fig. 13. TSF noise of two covert payloads when we gradually decrease n/m ratio. Each row belongs to one sample file. The left-most plots are isolated noise which means $m = n$. After we get to around 0.40 for the ratio, we start to see noise patterns that look sufficiently close to natural TSF noise of regular access points.

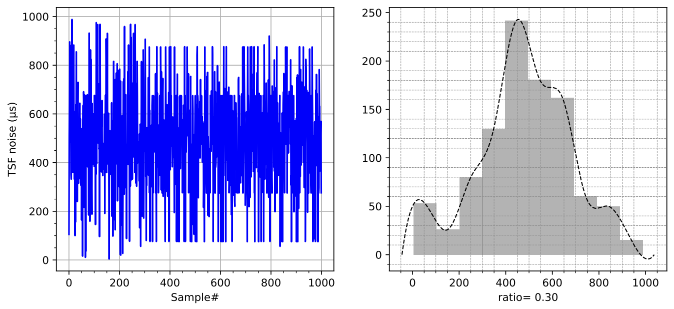


Fig. 14. TSF noise for transmission of a captured image by a surveillance camera with 1000x1000 resolution. In this figure, 1000 samples have been shown, each sample including one captured beacon of a random CHAOS access point. CAP noise normalized using a normal distribution function with $\mu = 500$ and $\sigma = 150$.

be transmitted from one or multiple interfaces depending on how we configure the system. As mentioned before, CHAOS can use a separate physical NIC for each AP to ensure physical isolation between the access points. This will effectively create real separate stations which are centrally managed by the same code. As a result, any physical layer audit would associate CHAOS APs to different physical stations.

C. Mitigation

Here we discuss mitigation in terms of stopping or interrupting the system and recovery of covert data.

1) Stopping CHAOS: There are two general ways to stop CHAOS:

- Jamming: The first approach is jamming 802.11 radio communication in the environment which effectively makes WiFi communication impossible. We have stated that CHAOS blends with the existing traffic. So we assume that WiFi communication is allowed and already exists in the area.
- Identifying stations: The other way is finding the stations that take part in CHAOS covert communication. We explained previously that finding the receivers is not possible since they do not transmit any frames related to this covert communication. To find the transmitter, an adversary has to first find out which beacons are being used by CHAOS and then try to identify the location of the transmitter (for example, by triangulating) and then if it is possible, physically shutdown the transmitter. We explained in detail, that identification of even one CHAOS AP is extremely challenging even with careful statistical analysis.

2) Forging CHAOS frames: If an adversary can identify some of CHAOS access points and then transmits beacons with the MAC addresses of those access points, they might be able to interrupt normal functionality of the system. However, the adversary has to transmit timestamps that are consistent with the permutation parameters and even then CHAOS receivers are able to filter out duplicate forged frames based on the signal power and other properties that make these frames distinguishable from CHAOS frames that the receivers are familiar with. Addressing all potential hurdles to successfully forge CHAOS frames will be extremely challenging for an adversary.

Also note that the beacons that CHAOS generates do not have any identifying information. In other words, a beacon generated by CHAOS looks *exactly* like a beacon sent by a legitimate access point.

3) Recovery of covert payload: In a covert channel, only the parties that take part in the communication should be aware of the existence of the covert data in the first place. But here we want to discuss the overall possibility of recovering the covert data by other stations with the assumption that they are aware that CHAOS is being used to transmit covert data.

Obviously, the covert payload can be encrypted before transmission. So data recovery by an adversary is not a big concern in CHAOS. That said, even in the case of plain (unencrypted) covert data, we want to make sure that other stations cannot decode it. Normally, only CHAOS receivers are able to decode and recover covert data sent by a CHAOS transmitter for three reasons:

- It is not clear for an adversary *which* access points are transmitting covert data using TSF noise.
- The adversary does not know the permutation parameters used by a CHAOS transmitter.
- Also, the adversary does not have the TXAP rotation key which is only shared between CHAOS stations.

Note that, even if an adversary can figure out all the access points that are advertised by CHAOS and even if they figure out the correct permutation parameters and the correct order-to-permutation assignments, without knowing the TXAP rotation key, it is not feasible to recover the covert data. For example, if the covert payload is only 6 data units long, there are $6^{\binom{m}{n}}$ different possible combinations for the covert payload to try. For $(m = 20, n = 6)$ there will be $6^{27907200}$. If the size of covert payload increases to 100 units, to reconstruct the whole payload[8] we will have $100^{27907200}$ combinations to try if TXAP key is not known. It is worth noting that, the adversary cannot even calculate all the combinations at once, but they have to wait one TBTT interval to the get next burst of beacons. Given the cost and the requirements, it is safe to assume that for an adversary, or basically any other station that is not a CHAOS transmitter or receiver, data recovery is not possible within a reasonable time and resources.

D. CHAOS Noise Resistance

A straightforward idea to make exploiting TSF harder is trying to make CHAOS frames distinguishable from other frames by increasing noise in TSF-based timestamps in regular access points. For example, by changing the implementation of the standard at firmware level. However, this will not help stop this type of covert communication.

CHAOS imitates the existing noise in the environment. The timestamp values in beacon frames generated by the same device using the same timer, relate to each other not an external timer. We do not compare them against a reference clock. It means that the accuracy of the TSF counter in access points does not affect CHAOS. Remember Equation 1. Both ts_i and ts_{i+1} are from the same access point. If we reduce the accuracy of the counter by increasing the clock drift, beacons should still be sent in TBTT intervals and the interval is measured through timestamp value which is written in beacon and probe response frames. The same equation still holds true. In other words, it does not matter the number that we see in the timestamp is accurate or not in terms of the *real* time that has passed during the interval. The reason that TSF noise exists is *not* TSF counter inaccuracy.

The other important point is that the TSF noise itself is a variable and random value. Whatever distribution this noise follows, we can make our CAP normalized noise to adapt to it the same way we did in the tests that we discussed in this section. While TSF noise is normally less than a millisecond, CHAOS can easily adapt to other levels of noise thresholds. Neither, changing the TBTT nor the maximum TSF noise can interfere with the functionality of CHAOS.

[8]An adversary might try to reconstruct only a small part of the covert data to figure out the rotation key, but there will still be too many combinations to consider.

E. Adversarial Signal Power Measurement

In the previous sections, we mentioned that an adversary might attempt to compare the signal power of beacons in the environment to find a pattern to identify what beacons might be generated by the same source. However, as discussed earlier, this can be mitigated in CHAOS if we use different network cards to transmit CHAOS beacons. Each AP can be assigned to a different NIC (for example, a USB WiFi NIC) placed in a different location. This effectively creates unpredictable variances between the radio frames used in CHAOS. As a result, the frames will not show similar radio characteristics in any location that could be observed by an adversarial monitor.

V. Discussion

TSF, what is special about it?

While timestamp fields in different network layers can be used for covert data transmission, TSF specifically has two interesting characteristics that make it unique and the reason that we chose it to build CHAOS. First, beacons have to be sent periodically within relatively short intervals. This means that whenever a transmitter wants to send covert data using TSF noise, there is a plausible reason in the first place to transmit a standard frame which has this type of noise to exploit. In other words, all access points are required by the standard to transmit beacon frames, many times per second, under any circumstances.

Second, TSF timestamp consists of two mathematical components: $TBTT + \varepsilon$. The first component is a predictable value and therefore the second component can be easily recovered at the receiver side. Also, the second component is a naturally variable value (noise) that changes in each transmission. This situation creates an excellent opportunity to encode and decode covert data in a manner that the standard's requirement is fulfilled while secret data is hidden in the noise at the same time without diverging from normal AP behavior. Note that the resolutions of the first and the second component match which makes the noise *measurable* in the timestamp field itself and makes exploitation more straightforward.

These characteristics do not exist in other types of timestamps (e.g., TCP timestamps). Also, remember that TSF timestamp is a mandatory part of beacons as stated in 802.11 standard. They cannot be disabled by users unlike some other types of timestamps in other protocols. As we explain in the following, even doing so is not a reasonable solution.

The unique nature of TSF timestamps, coupled with the density of access points (and their advertisement frames) in urban areas, makes TSF an outstanding target for stealth communication.

What can be done to prevent TSF exploitation?

The most straightforward strategy to prevent this is disabling or removing timestamps from beacons and probes. This is of course not possible in the current design of 802.11 standard as doing so will disable station-level time synchronization which is crucial for low-level radio and frame scheduling and consequently WiFi communication will entirely fall apart. Actually, not only TSF is not going to disappear, but also new applications are introduced for it [7]. Interestingly, even if this solution was implemented and timestamp field was removed, we could make a similar covert channel by using similar timing patterns in the IFDs themselves instead of a timestamp field. Because the beacons would be still transmitted in short intervals and the constant flow of a timing pattern that can be used for covert communication is guaranteed. The order of the beacons and the sequence of IFDs can create the permutations we need to encode secret bits. Having the timestamp field makes the covert communication easier and more reliable but the same goal can be theoretically achieved without it.

The other solution will be adding some randomness to TBTT to make the first component above, less predictable. Again, this will not help to resolve the issue. We can trivially measure an average minimum TBTT and substitute it with TBTT in Equation 1, and then increase the TSF noise by either increasing L or DS. The same results will be achieved. Note that in Equation 1, we can use negative values for ε and calculate the absolute value of the difference between two consecutive timestamps to get the deviation from the nominal value as the noise level. In other words, the generated noise can be subtracted from the timestamp value instead of being added to it.

Interestingly, currently in normal access points the observed TBTT (the nominal interval without the TSF noise) is not necessarily equal 100TUs between every two beacons. The actual interval can be a little shorter or longer based on the exact situation of medium and scheduling at the moment that beacons are transmitted. It means that there is already some level of unpredictable randomness. But even increasing this randomness will not stop CHAOS.

At the time being, probably the only effective solution is changing the design of time synchronization in the protocol so that the synchronization process does not have the side effects that we exploited in this paper. This might come at the cost of reduced WiFi throughput depending on how the synchronization takes place. Specially if the periodic and frequent transmission of beacons that is currently mandated in the standard is disabled. On the other hand, it is possible to define new synchronization strategies only for sensitive use cases and network environments that require exceptional levels of security and privacy to limit issues like reduced bandwidth and throughput to these specific use cases.

Legitimate Uses

While at the surface, a covert channel might have an adversarial nature, it can be used for the same purposes that non-covert communication is used for. A covert channel can add an extra layer of privacy and protection in cases that standard communication protocols are not sufficient to transmit important information in situations that high levels of secrecy are required. Covert channels can improve privacy of communication. The stealthiness that a covert channel provides can have vital advantages for some specific use cases.

In Section IV we demonstrated the robustness of CHAOS to reliably exchange covert data without being exposed to an adversary. We believe the strategy that we presented in this paper can help provide safer communication of sensitive information in cases that normal communication does not meet the privacy expectations of the user or the data.

VI. Related Work

Many covert channels have been designed for different protocols at different layers [34], [40], [4], [42], [9], [12], [18], [23], [31], [15], [14], [41]. Most previous works discuss covert communication in technologies other than WiFi.

Covert channels in 802.11 networks have been addressed in two generic topics: physical layer and MAC layer.

Physical layer covert channels in wireless technologies usually use phase, amplitude and frequency manipulation [11], [36], [13] or use IFDs to encode covert data as timing patterns [16], [30], [20], [10]. [29] explains that most physical layer covert channels can be detected through careful statistical analysis of different aspects of carrier signals at physical layer.

[33] creates a single access point on a Zynq board and uses two timing thresholds for mapping a beacon frame to either a zero or a one symbol. Specific sequences of the recorded symbols are then translated to a data bit of one or zero. The proposed approach is substantially limited and ends up with a data rate of a few bits per second at best. The problem of missed frames is not fully addressed in the paper and there is not any discussion about adversarial noise injection and the effect of changing TBTT on their channel. More importantly, some of the given beacon statistical reports (which establishes the basis of their covert model) do not match with our real world observations.

[32] uses the order of the medium access of two stations to transmit covert bits; if station A transmits before station B it means zero and otherwise one. They also discuss using frame orders in general as a means for covert transmission. [37] makes a covert channel using the *supported rates* field of probe request frames. [38] uses the transmitter MAC address of probe request frames to send covert payloads. Multiple other works have used other different aspects of 802.11 frames to implement covert channels [6], [19], [27]. The subject of TSF noise has not been addressed before, on top the fact that most prior work have limited uses and there is usually no sufficient discussion of mitigation techniques.

The subject of correlation attack has been covered using timing and flow analysis of packets as well as application layer exploitation [24], [28], [25], [39], [35]. Mitigating correlation attacks depends on the specific attack techniques and threat models in use. Techniques like introducing some random delay and increasing noise through inserting extra packets in the network have been studied before. Unfortunately there is no generic solution to prevent these attacks against low-latency anonymity networks. For the first time, we investigated the side effects and uses of TSF in 802.11 networks for mounting correlation attacks.

CHAOS, to the best of our knowledge, is the first work that discusses in-depth exploitation of TSF counters.

VII. Conclusion

Time Synchronization Function within 802.11 networks presents a unique opportunity for covert communication. The variable noise in TSF-based timestamps can be exploited to encode secret messages between WiFi stations. This type of noise is also influenced by frame load, opening up a potential for fingerprinting users by correlating the induced load with fluctuations observed in the TSF noise pattern. In this paper, we delved deep into the unintended consequences of TSF design and the exploitation of these side effects. We demonstrated that it is possible to use TSF timestamps to reliably broadcast covert messages. Additionally, we showed that we can directly manipulate the noise patterns by altering the traffic that an access point must manage.

References

[1] "Cisco annual internet report," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] "Current iot forecast highlights," https://transformainsights.com/research/forecast/highlights.

[3] "Wi-fi certified 6," https://www.wi-fi.org/discover-wi-fi/wi-fi-certified-6.

[4] C. Abad, "Ip checksum covert channels and selected hash collision," *USA, University of California*, 2001.

[5] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, ser. MobiCom '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 116–127.

[6] T. E. Calhoun, X. Cao, Y. Li, and R. Beyah, "An 802.11 mac layer covert channel," 2012.

[7] P. Chen and Z. Yang, "Understanding precision time protocol in today's Wi-Fi networks: A measurement study," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021.

[8] J. Classen, M. Schulz, and M. Hollick, "Practical covert channels for wifi systems," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015.

[9] D. M. Dakhane and P. R. Deshmukh, "Active warden for tcp sequence number base covert channel," in *2015 International Conference on Pervasive Computing (ICPC)*.

[10] A. Dutta, D. Saha, D. Grunwald, and D. Sicker, "Secret agent radio: Covert communication through dirty constellations," 2013.

[11] S. D'Oro, F. Restuccia, and T. Melodia, "Hiding data in plain sight: Undetectable wireless communications through pseudo-noise asymmetric shift keying," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[12] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through tcp timestamps." Springer-Verlag, 2002.

[13] S. Grabski and K. Szczypiorski, "Steganography in ofdm symbols of fast ieee 802.11n networks," in *2013 IEEE Security and Privacy Workshops*, 2013.

[14] M. Guri, A. Kachlon, O. Hasson, G. Kedma, Y. Mirsky, and Y. Elovici, "GSMem: Data exfiltration from Air-Gapped computers over GSM frequencies," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[15] J.-W. Ho, "Covert channel establishment through the dynamic adaptation of the sequential probability ratio test to sensor data in iot," *IEEE Access*, 2019.

[16] R. Holloway and R. Beyah, "Covert dcf: A dcf-based covert timing channel in 802.11 networks," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, 2011.

[17] IEEE, "Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11," https://standards.ieee.org/ieee/802.11-2020_Cor_1/10836/.

[18] E. Jones, O. Le Moigne, and J.-M. Robert, "Ip traceback solutions based on time to live covert channel," in *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004) (IEEE Cat. No.04EX955)*, 2004.

[19] T. Kim and W. Lee, "Channel independent wi-fi backscatter networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[20] N. Kiyavash, F. Koushanfar, T. P. Coleman, and M. Rodrigues, "A timing channel spyware for the csma/ca protocol," *IEEE Transactions on Information Forensics and Security*, 2013.

[21] Y. Lin, Y. Gao, B. Li, and W. Dong, "Accurate and robust rogue access point detection with client-agnostic wireless fingerprinting," in *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2020.

[22] P. Liu, P. Yang, W.-Z. Song, Y. Yan, and X.-Y. Li, "Real-time identification of rogue wifi connections using environment-independent physical features," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[23] X. Luo, P. Zhou, E. W. W. Chan, R. K. C. Chang, and W. Lee, "A combinatorial approach to network covert communications with applications in web leaks," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, 2011.

[24] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew." Association for Computing Machinery, 2006.

[25] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," 2018.

[26] N. T. Nguyen, G. Zheng, Z. Han, and R. Zheng, "Device fingerprinting to enhance wireless security using nonparametric bayesian method," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 1404–1412.

[27] R. Ogen, K. Zvi, O. Shwartz, and Y. Oren, "Sensorless, permissionless information exfiltration with Wi-Fi Micro-Jamming," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[28] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[29] H. Park, W. Jang, J. Sung, H. Roh, and W. Lee, "Combating adversarial covert channels in wi-fi networks," *IEEE Access*, vol. 10, 2022.

[30] S. V. Radhakrishnan, A. Selcuk Uluagac, and R. Beyah, "Realizing an 802.11-based covert timing channel using off-the-shelf wireless cards," in *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013.

[31] C. H. Rowland, "Covert channels in the tcp/ip protocol suite," *First Monday*, 1997.

[32] K. Sawicki, G. Bieszczad, and Z. Piotrowski, "Stegoframeorder—mac layer covert network channel for wireless ieee 802.11 networks," *Sensors*, vol. 21, 2021.

[33] H. Seong, I. Kim, Y. Jeon, M.-K. Oh, S. Lee, and D. Choi, "Practical covert wireless unidirectional communication in ieee 802.11 environment," *IEEE Internet of Things Journal*, 2023.

[34] G. Shah and A. Molina, "Keyboards and covert channels," in *15th USENIX Security Symposium (USENIX Security 06)*. Vancouver, B.C. Canada: USENIX Association, 2006.

[35] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," ser. CCS '18. Association for Computing Machinery, 2018.

[36] K. S. Subramani, N. Helal, A. Antonopoulos, A. Nosratinia, and Y. Makris, "Amplitude-modulating analog/rf hardware trojans in wireless networks: Risks and remedies," *IEEE Transactions on Information Forensics and Security*, 2020.

[37] G. Teca and M. Natkaniec, "An ieee 802.11 mac layer covert channel based on supported rates," *International Journal of Electronics and Telecommunications*, 2023.

[38] ——, "A novel covert channel for ieee 802.11 networks utilizing mac address randomization," *Applied Sciences*, vol. 13, 2023.

[39] R. Wails, Y. Sun, A. Johnson, M. Chiang, and P. Mittal, "Tempest: Temporal dynamics in anonymity systems," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, 2018.

[40] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays." New York, NY, USA: Association for Computing Machinery, 2003.

[41] Z. Yang, Q. Huang, and Q. Zhang, "Nicscatter: Backscatter as a covert channel in mobile devices." Association for Computing Machinery, 2017.

[42] X. Zhang, Y.-A. Tan, C. Liang, Y. Li, and J. Li, "A covert channel over volte via adjusting silence periods," *IEEE Access*, 2018.

Appendix

A. Beyond Covert Communication

A Correlation Attack

During the course of our tests, we observed that the distribution of TSF noise *significantly* changes when the WiFi router is under heavy load. When we increased the number of frames that the access point has to manage, the distribution of the noise, surprisingly, tended to get close to a uniform distribution instead of a semi-normal distribution. While the exact reason of why this happens is not completely clear, we suspect two potential factors that might cause this behavior. First, when the number of TX/RX frames increases, the medium gets busier and the router has to queue the beacon frames more often compared to when the medium is less frequently used for frame transmission. This will change the amount of time that the access point pauses before sending out a beacon and consequently intervals between beacons change to both shorter and longer values. Second, the load increase can temporarily change the temperature of the chipset and result in more unpredictable environmental noise.

Note that the increased load of one WAN W_i can affect TSF noise of other access points from other WANs on the same channel if they are close enough to receive the signals from W_i with sufficient power. The effect of induced load from other WANs depends on their distances. Usually, TSF noise is not affected significantly by the frames transmitted from stations belonging to other WANs. The CSMA/CA implementation in WiFi chips ignores many frames coming from unrelated stations with weak signals.

We have tested multiple access points made by different manufacturers and all of them show a similar behavior in response to changing load in both 2.4GHz, and 5GHz bands.

This will have an important side effect: an attacker can affect the access point's traffic at one side through forcing one of the stations in the BSS to increase its load, and then confirm that the changes are reflected on the TSF noise at the other side to find which physical access point the targeted client is connected to. In other words, we can reliably implement an end-to-end confirmation attack.

When load increases, TSF values tend to deviate more aggressively from their mean. In Appendix III-D, we have reported a detailed analysis of the impact of changing traffic on TSF distribution in multiple WiFi routers from different brands. Since changing the access point load immediately affects the TSF noise, sampling the beacon frames for a short time, like ten seconds, is enough for mathematical confirmation of the correlated changes of the TSF deviation with a high success chance.

The Attack Scenario

Assume that an attacker wants to find out which physical access point a targeted WiFi user is connected to. The attacker is able to see WiFi frames of a set of access points like $S = \{AP_1, ..., AP_p\}$ and knows that the victim is connected to one of these access points (like AP_i) but does not know which one. In this situation, the attacker can follow these steps to identify AP_i with high confidence within minutes:

- The attacker increases the load of the victim's network by having the victim download a file (or an online content) which is large enough to take at least a couple of seconds[9].
- The attacker first confirms that the file is being downloaded, at the server side, by the victim.
- Immediately after the download starts, the attacker observes the deviation of TSF values of beacon frames transmitted by all the access points in S.
- The attacker notices that TSF values of AP_i show the expected deviation precisely during the download time.
- The attacker can choose to repeat the download stage a couple of times to rule out possible false positives.
- Eventually, the access point that the victim is connected to (AP_i) and his approximate physical location is compromised.

Executing all these steps can take a couple of minutes. Note that the user does not even have to voluntarily download the file. It can happen indirectly through a webpage that the user visits.

If none of the access points in S shows the expected changes in its TSF values, the attacker can confidently rule out the current members of S and switch to another set. This is the main limitation for the attacker, that eventually the correct access point must be one of the access points in S to be correlated with the traffic manipulation and complete the identification process. This attack can be used to find a user that the adversary knows resides in a specific neighborhood but does not know in which house. Or in the case of a much more resourceful adversary, the same technique can be applied to a larger geographical area like a city.

Here we describe a real-world example of how it is possible for an adversary to exploit the dependence of noise distribution on the AP load and use TSF noise to distinguish between the access points that users are associated with. In this section we use an experiment to execute a confirmation attack to deanonymize one of our systems when it is connected to TOR. The setup is as follows.

We target 5 access points and fingerprint TSF noise throughout the test. $S = \{AP_1, .., AP_5\}$. One or more stations are connected to each access point. The stations are either on separate channels, or they are far enough from each other that transmitted frames from one AP do not cause significant signal collision on the others[10]. One of the stations connects to TOR network using TOR-Browser-Bundle and its default

[9]We have run this attack with a 10-second download window.

[10]A distance of 20 meters is enough. Access points are normally further away from each other in residential areas.

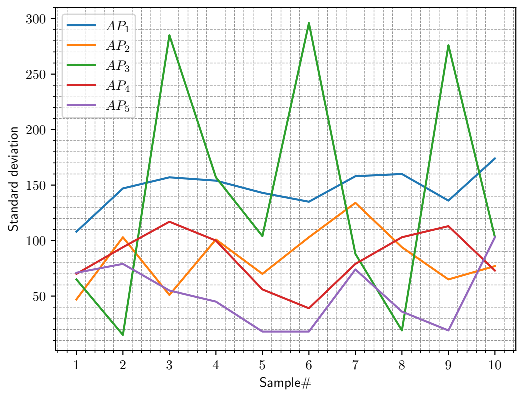


Fig. 15. The plot of standard deviation of TSF noise for the tested access points. The download task happened at sample numbers 3, 6 and 9. The only access point that shows abnormal changes at these sample points is AP_3.

configuration. Any external entity that makes the user of the station to change his network activity, can induce changes on the TSF noise pattern. For example, we instruct the user of this station to visit a specific address through TOR browser. The webpage contains a piece of js code in the website that periodically downloads a set of large images which takes around 10 seconds to complete.

At the same time we record the beacon frames from the access points in S and observe the TSF noise patterns of each access point. We sample TSF noise 10 times during the test. In each sample, we captured 100 frames. Three of the samples belong to the time that the TOR user is performing the download task. As Figure 15 shows, the induced traffic by the remote server is correlated to AP_3 as its noise distribution alters exactly based on the periodic download task. Similar steps can be carried out to identify the correct access point with a larger set S.

The test successfully confirms that the user that visited the remote webpage is connected to AP_3.

B. The Effect of Frame Load on TSF

We investigated how TSF noise patterns alter when we change traffic of an access point. In this section we have added the results of three of the access points that we have tested. A single band access point made by TP-LINK and two dual band access points made by NETGEAR and Xfinitiy. In Figures 16 through 20, in each figure, the first eight plots show TSF noise when the access point is in a semi-idle state in which the connected clients do not perform any network-intensive task and the second eight plots show noise distribution when exactly one of the clients uses all its available bandwidth to download a file (top and bottom rows respectively in Figures 17 to 20). We have taken 100 samples of beacons of these access points in each test. The standard deviation of each distribution is written in the corresponding graph.

In summary, the standard deviation in these access points changes the way it is numerically explained in Table III. We see more than 100% increase in the average deviation when only one client in the WAN starts using up its available bandwidth. This shows a strong dependence on load. This also makes it easier for CHAOS to create a plausible TSF pattern if we reduce m and use less normalized noise for CHAOS access points. Ergo, fewer beacons can potentially be used in each burst while we maintain undetectability of the system in most normal WiFi environments.

TABLE III
AVERAGE VALUES OF STANDARD DEVIATION OF NOISE DISTRIBUTION OF THE ABOVE ACCESS POINTS.

Access Point	Band	Idle	Under Load	Growth
TP-LINK	2.4GHz	116	280	141%
NETGEAR	2.4GHz	193	293	52%
NETGEAR	5GHz	66	279	323%
xFi	2.4GHz	118	283	140%
xFi	5GHz	178	267	50%
Average		134	280	141%

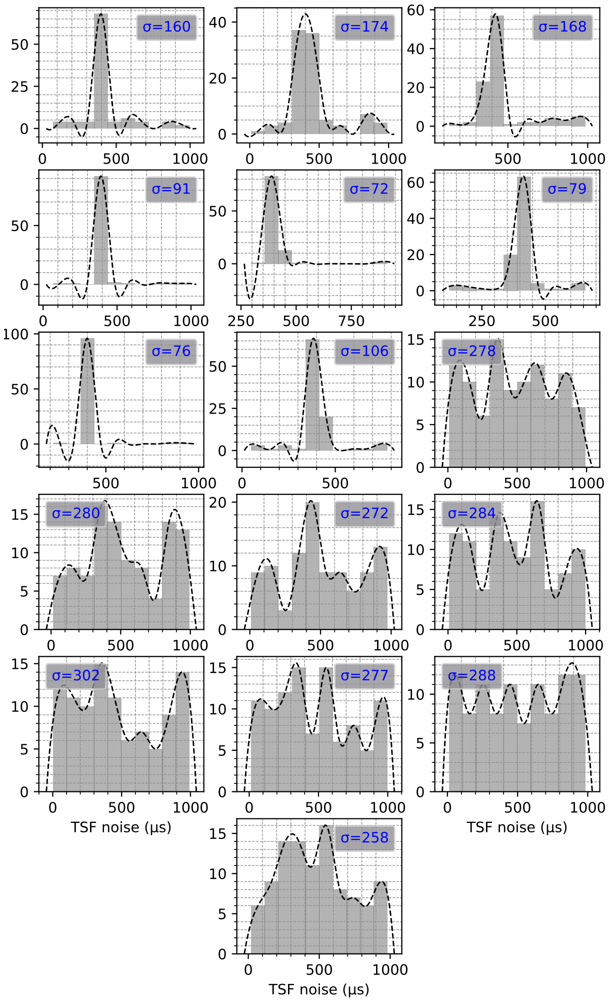


Fig. 16. TP-LINK (2.4GHz)

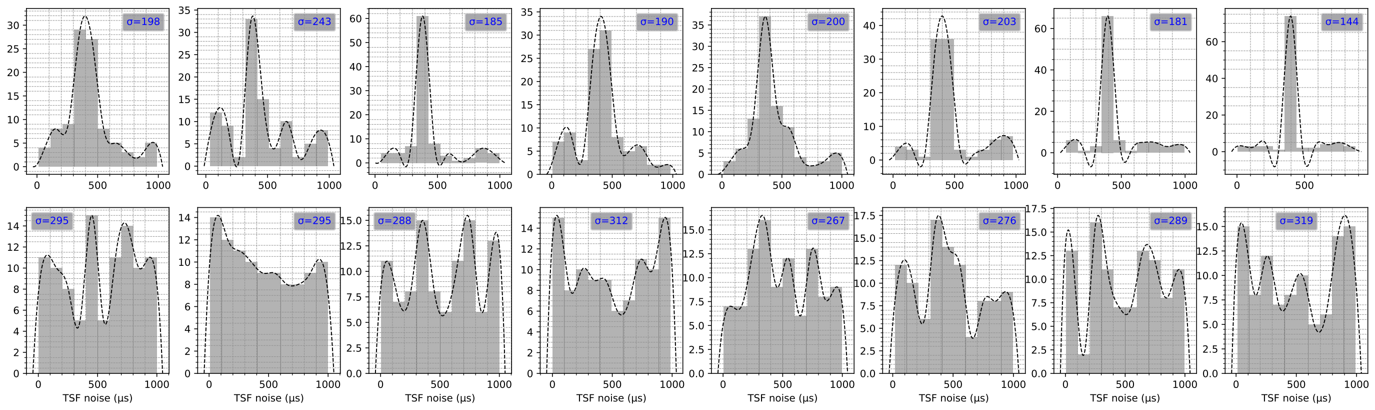


Fig. 17. NETGEAR (2.4GHz)

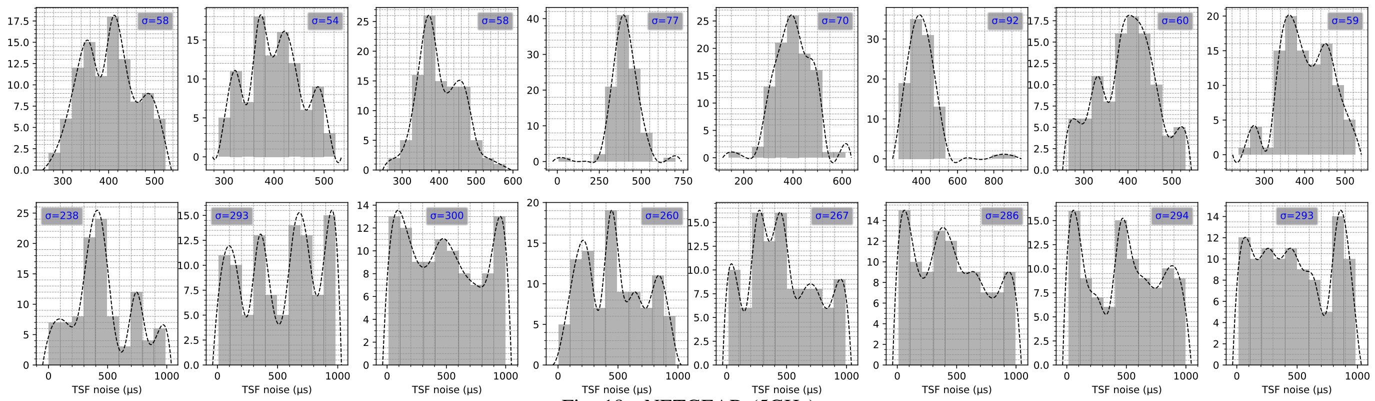


Fig. 18. NETGEAR (5GHz)

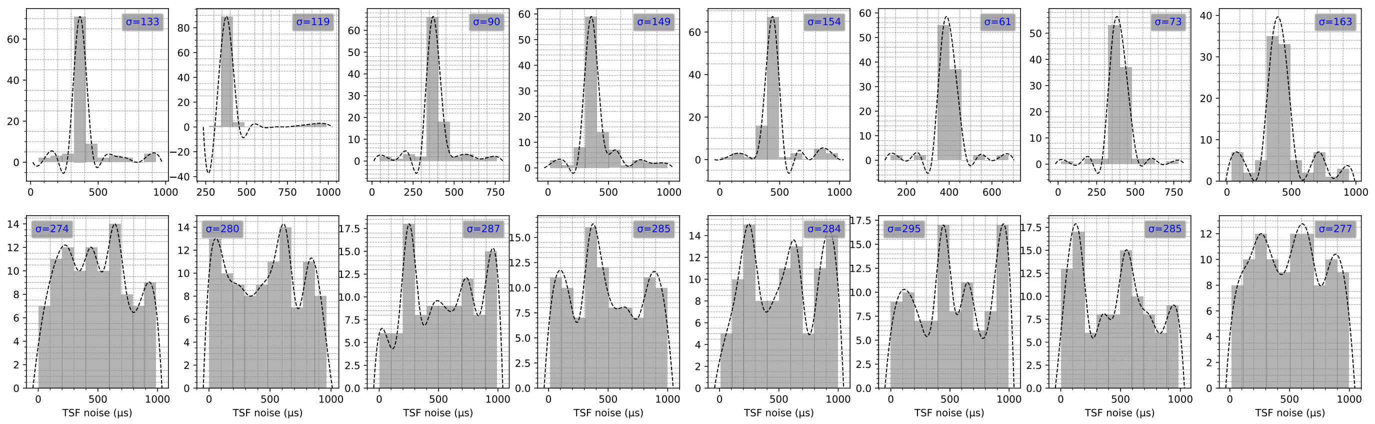


Fig. 19. Xfinity xFi Gateway (2.4GHz)

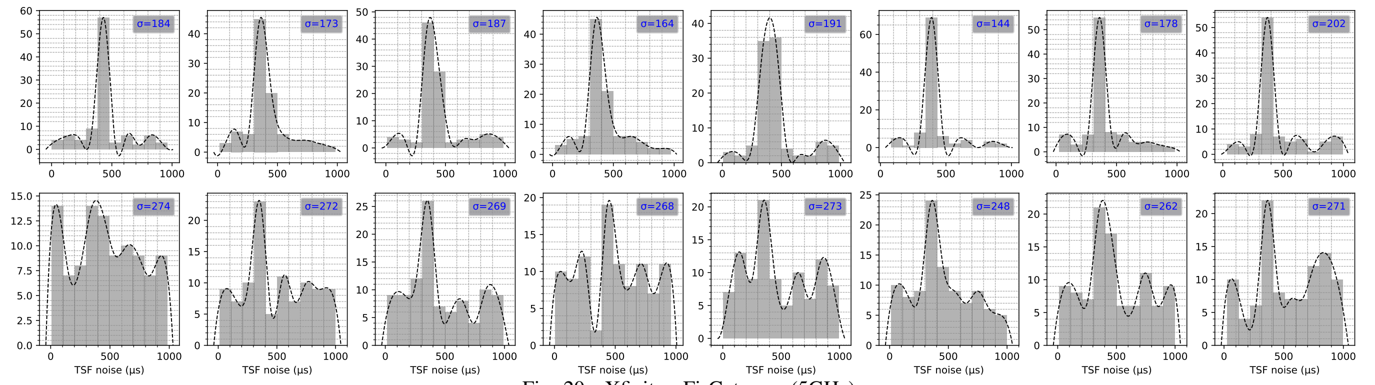


Fig. 20. Xfinity xFi Gateway (5GHz)