A Framework for MIMO-based Packet Header Obfuscation

Yue Cao*, Ahmed O. F. Atya*, Shailendra Singh*, Zhiyun Qian*,
Srikanth V. Krishnamurthy*, Thomas La Porta†, Prashant Krishnamurthy⋆, Lisa Marvel‡,
*University of California, Riverside, †Penn State University, ‡US Army Research Laboratory, ⋆University of Pittsburgh
{ycao009, afath001, singhs, zhiyunq, krish}@cs.ucr.edu, tlp@cse.psu.edu, lisa.m.marvel.civ@mail.mil, prashant@mail.sis.pitt.edu

Eavesdroppers can exploit exposed packet headers towards attacks that profile clients and their data flows. In this paper, we propose FOG, a framework for effective header blinding using MIMO, to thwart eavesdroppers. FOG effectively tracks header bits as they traverse physical (PHY) layer sub-systems that perform functions like scrambling and interleaving. It combines multiple blinding signals for more effective and less predictable obfuscation, as compared to using a fixed blinding signal. We implement FOG on the WARP platform and demonstrate via extensive experiments that it yields better obfuscation than prior schemes that deploy full packet blinding. It causes a bit error rate (BER) of > 40 % at an eavesdropper if two blinding streams are sent during header transmissions. Furthermore, FOG incurs a very small throughput hit of $\approx$ 5 % with one blinding stream (and 9 % with two streams). Full packet blinding incurs much higher throughput hits (25 % with one stream and 50 % with two streams).

I. Introduction

Wireless links are susceptible to eavesdropping attacks. Recent standards like 802.11i and 802.11w propose to use link layer encryption, but still expose MAC layer addresses of both APs (access points) and clients. By just capturing the MAC addresses an attacker can analyze traffic to figure out how much data is being transferred between the AP and particular clients. This information can be used to perform various potent attacks. For instance, user input such as keystrokes to web applications can be inferred [1]. Jamming attacks can also target clients that are receiving heavy volumes of traffic [2].

In practice the problem is worse since 802.11w is not used in most scenarios (e.g., in airports or coffee shops). Thus, by intercepting control frames (e.g., ACKs), an attacker can now infer the amount of traffic exchanged with a particular server (in the wired network) since it can access the source and destination IP addresses, port numbers etc. One could conceivably encrypt the entire packet including the MAC layer header to prevent traffic analysis. However, as we discuss later this is hard to do in practice.

Use of MIMO to cope with eavesdropping: An alternative approach for thwarting eavesdroppers, is the use of an antenna array or MIMO (multiple-input multiple-output) for obfuscating data streams. As proposed in [3], a subset of the transmitter's degrees of freedom can be used to transmit *blinding* streams that interfere with a legitimate stream in the area around the transmission (and thus foil eavesdropping), except at the intended recipient.

Fundamental challenge and tradeoff: MIMO-based obfuscation does not guarantee that an attacker will be unable to decipher eavesdropped information. This is because while a blinding signal will effectively disturb the legitimate signals in most locations, there could be places where it does not. One way to cope with this is to use a plurality of blinding streams. However, as the number of blinding streams is increased for better obfuscation (as in [3]), the achievable throughput on the MIMO link decreases (since fewer legitimate streams can now be transmitted). In fact, even the use of a single continuous blinding stream decreases the throughput by $\frac{1}{N}$ where, N is the number of antennas at the sender.

Our observation: Given that most sensitive traffic is sent using TLS/SSL or in some cases link layer encryption, it makes sense to use MIMO to blind only the headers (instead of using a continuous blinding stream as in [3]). This can drastically reduce the performance penalties that are experienced with traditional MIMO-based obfuscation (e.g., [3]).

Challenges in realizing header blinding in practice: First, due to PHY layer functions such as scrambling and interleaving, symbols that correspond to the header bits are dispersed throughout a packet to be transmitted. Furthermore, convolution codes (commonly used) jointly encode header and payload bits. Thus, it is inevitable that symbols associated with the header are intertwined with some payload bits that will need to be blinded out as well. Correctly identifying the symbols to blind out is a challenge that has to be addressed.

Second, the header bits in the multiple streams that are to be transmitted simultaneously are not aligned. Thus, if the header bits in any stream are to be blinded, the transmissions of symbols from one or more of the other streams will need to be temporarily *suspended* during that time to allow for a blinding signal(s) to be transmitted instead. This leads to challenges relating to scheduling the blinding signals and providing information about suspensions to legitimate receivers.

Security issues with MIMO-based obfuscation: Recently, it was shown that if an eavesdropper knows some of the transmitted bits (e.g., MAC address of the AP), it can filter received data to extract the blinding signal [4]. Knowing the blinding signal allows the decoding of the remainder of the protected information. These attacks are called *plaintext attacks against physical layer security*. This is of concern since broadcast packets (e.g., for association) are sent in the open, and thus, reveal the AP's MAC address.

Contributions: In this paper, we build a practical protocol framework, *FOG*, for effectively performing header blinding to protect wireless transmissions from eavesdroppers. As discussed above, header blinding significantly improves throughput over traditional full packet blinding. To identify the header symbols, *FOG* uses tainting to track the header bits as they traverse the physical layer system (scrambler, convolutional coder and interleaver).

Instead of using a fixed stream with a single blinding signal, *FOG* schedules different blinding signals at finegrained intervals, achieving increased randomness. This makes it almost impossible for an eavesdropper to find locations where the combined effects of the blinding signals are ineffective. This also dramatically raises the bar against *plaintext attacks against physical layer security*; since the blinding signals change dynamically over small time intervals, it is computationally hard for an attacker to launch such an attack.

In summary, our contributions are as follows:

- We design *FOG* for effectively obfuscating wireless packet streams. *FOG* applies MIMO-based blinding that can be used in conjunction with encryption to effectively thwart eavesdroppers from performing traffic analysis on wireless links. *FOG* works with TLS, IPsec, or link layer encryption, obfuscating any headers transmitted in the open.
- We implement *FOG* on our WARP radio testbed [5]. We perform extensive experiments to show that *FOG* provides even better obfuscation than traditional full packet blinding (as in [3]). With two blinding streams, the fraction of locations where an eavesdropper experiences a 40 % bit error rate improves from almost 0 % to 100 %. Further, the throughput hit with *FOG* is much lower than with full packet blinding (5 % as compared to 25 % with the latter with one blinding stream).

II. Background

In this section, we describe how blinding (also called nulling) is achieved with multi-user MIMO (MU-MIMO). We consider the popular practical low complexity zero-forcing beamforming (ZFBF) [6]; ZFBF completely removes the interference among the MU-MIMO transmissions.

Notation: N is the number of transmit antennas at the AP. M is the number of concurrently served receivers (considered single antenna for ease of discussion but can be MIMO equipped). The row vector h_m is a $1 \times N$ vector, representing the channel state with respect to user m. Each element of h corresponds to the complex gain from one transmit antenna to the user. The matrix $H = [h_1; h_2; ..; h_M]$ is the channel matrix of dimension $M \times N$. The column vector w_m is an $N \times 1$ beamforming weight vector for user m. Each element of w corresponds to the complex exponential weighting used by each antenna during transmission to that user. The matrix $W = [w_1, w_2 ... w_m]$ is the weight matrix of dimension $N \times M$. If A streams are used for blinding, $N - A$ streams are available for legitimate communications.

Zero Forcing Beam-forming (ZFBF): ZFBF enables a transmitter (AP) to construct multiple spatial streams and transmit them to multiple users in parallel. The channel state information (CSI), h_k, is obtained for each user k, using pilots and a corresponding w_k is computed. A composite data stream (consisting of streams destined for different users) is then multiplied by W and transmitted using the antenna array. ZFBF selects weights that cause zero inter-user interference. In [7], it is shown that the optimal W, satisfying the zero inter-stream interference condition, is the pseudo-inverse of H, i.e., $W = H^{\dagger} = H^{*}(HH^{*})^{-1}$. The above matrix multiplication implicitly requires that the maximum number of concurrent spatial streams M, to be less than or equal to the number of transmit antennas, N. The CSI from each user (the h vector), can be fed back via the RTS/CTS in compliance with 802.11ac.

Blinding Process: Blinding signals must be orthogonal to the intended signals (so that they are not affected) and are transmitted concurrently with the intended receivers' signals by the ZFBF enabled transmitter. Together, the intended and the blinding streams should not be $> N$. For the blinding streams, $\hat{h}$ vectors that are orthogonal to the intended receivers' h vectors are computed. The precoding weight matrix W is then constructed by considering the blinding streams also as streams that are to be transmitted. To create the h vectors for a blinding stream, the Gram-Schmidt orthogonalization process is used (as in [3]). Let us assume that the first $N - A$ streams are designated to intended receivers, without loss of generality. The CSI $h_1,\ h_2, \ldots h_{N-A}$, are obtained and a matrix $\tilde{H}$ is created in the same way as H was created with traditional ZFBF. Here, we first create an Identity matrix I of size $N \times N$ and truncate it to a matrix $\hat{I}$ of size $(N-A) \times N$. $\hat{I}$ is now concatenated with the $\tilde{H}$ matrix to create a preliminary H matrix. This preliminary H matrix, is passed to the Gram-Schmidt process and the matrix, $\hat{H}$ is constructed, as follows:

$$\hat{h_k} = \begin{cases} h_k & \text{if } k = 1 \\ h_k - \sum_{j=1}^{k-1} \frac{\langle h_k, \hat{h_j} \rangle}{\|\hat{h_j}\|^2} \hat{h_j} & \text{if } k > 1 \end{cases}$$

where, k is the index of a vector and $< h_k, \hat{h_j} >$ is the dot product of h_k and $\hat{h_j}$. A corresponding precoding matrix $\hat{W}$ is then computed. The intended streams along with blinding streams (could be any random stream of bits), are then weighted using $\hat{W}$ and transmitted. Note that the process is similar for other combinations of legitimate streams (when they are not the first $N - A$ streams).

III. Motivation

Motivating Study: To understand the extent to which information is exposed, we perform a measurement study where we collect WiFi packets in a public coffee shop for 30 minutes using WireShark on Ubuntu 14.10 with Linux 3.16 (our office of research integrity indicated that an IRB was not required). We capture approximately 1.8 million packets, and identify more than 1900 destination IP addresses that clients were communicating with. Our results (details omitted due to space constraints) indicate that even an unsophisticated attacker can gather a lot of information since control and management packets are often transmitted unencrypted (even if link layer encryption is used for data packets). It has been previously shown that users can be identified and linked with confidential information such as their location history using such management information that is sent in the clear [8]. Further, an attacker, by observing the sequence of packets within a session, can tease out sensitive information about their contents. For example, patterns of packet sizes and timings are sometimes sufficient to identify the keystrokes [1], the web pages viewed [9], the languages spoken [10], and the apps used [11] by the user. In the best case, even with link layer encryption of all unicast packets (including control packets), the MAC addresses and CCMP (Counter Mode Cipher Block Chaining Message Authentication Code Protocol) header of clients are exposed. One could thus analyze the traffic (e.g., packet sizes, download patterns) that a client receives.

Encrypting headers is hard: It is hard with current 802.11 standards to fully encrypt the packet headers. In particular, CCMP encryption uses the AES cipher as its building block. AES uses counter values which the receiver derives from the CCMP header to perform decryption. If there are packet losses, it cannot pre-determine the contents in forthcoming headers to pre-compute the encrypted header values. Thus, the CCMP header has to be sent as cleartext. However, this header can form the basis for side-channel attacks (an example is in [1]). In addition, sending the CCMP header as cleartext can lead to what is called a "time memory trade-off pre-computation attack," which provides a shortcut for a brute force attack to get the encryption key (details in [12]). To encrypt the CCMP header, one needs another CCMP header in cleartext (therefore a chicken-and-egg problem). Thus, we argue that blinding headers with MIMO would be an effective countermeasure.

Prior work: In [13], *SliFi*, a system that tries to overcome the problem of encrypting everything including headers is proposed. Each receiver tries to pre-compute the encrypted headers and store them in a hash table. When packets are received, the hash of the header is computed and checked against table entries. These hash table checks may involve higher layer operations. While this might not degrade throughput, it could increase energy drain in wireless clients. We consider *FOG* to be an alternative PHY layer solution based on MIMO. Note that both *SliFi* and *FOG* require changes to both the APs and mobile clients (discussed later for *FOG*).

IV. System model and assumptions

As in 802.11ac, all broadcast frames or packets are sent using omnidirectional transmissions [14] to achieve maximum coverage. We assume a secure authentication scheme as described in the IEEE 802.11w standard. After authentication, a secret key is established between the AP and the client using methods described in the 802.11w standard [15].

CSI Exchange: The channel sounding process is used to facilitate the exchange of channel state information (CSI) between each client and the AP [14]. This process begins when the AP broadcasts a Null Data Packet (NDP) announcement frame. A multi-user NDP announcement frame includes multiple client information records, one for each client. Each record is encrypted using the secret key shared with the associated specific client. Following this, the AP broadcasts an NDP frame (see [14] for details). Each client then uses specific training fields in the NDP frame to compute its CSI matrix. It then sends the CSI matrix via a "Compressed Beamforming Action frame" to the AP. This frame is also encrypted using the secret key that is shared by the client with the AP.

What is protected and what is not: *FOG* uses 802.11w to encrypt management frames like NDP. Link layer encryption could be used (optionally) to protect data frames; in such cases *FOG* uses blinding to only protect the MAC and CCMP headers. If only higher layer encryption (e.g., TLS) is used, *FOG* blinds out the relevant higher layer headers as well.

Beacon frames and the association requests are not protected and can reveal MAC addresses of the AP and clients. Thus, *FOG* does not try to hide the identities of the clients that are connected to an AP. Its primary objective is to deter eavesdropping attackers from performing traffic analysis attacks to profile the associated clients. Each frame in 802.11ac (or similar systems) contains known preambles. These are transmitted without any beamforming to allow devices to synchronize and identify the beginning of a packet [14]. These preambles cannot be encrypted or blinded since at this point, the sender and receiver are yet to synchronize with each other; *FOG* does not hide preamble transmissions (they do not yield receiver specific information to an eavesdropper).

Downlink versus Uplink: *FOG* hides transmission patterns to prevent traffic analysis. It is agnostic to whether the transmissions are on the downlink or uplink, as long as the transmitter is equipped with MIMO. Today, most commodity APs are MIMO based. Recent smartphones and smaller client devices are also 802.11 ac capable (e.g., iPhone6 [16]). If a client is not MIMO based, it could encrypt everything (including the MAC header) on the uplink. The AP would then decrypt all received packets prior to processing. Since the AP is typically connected to a power outlet, this process should not adversely impact communications except for slight increases in delay. This approach however, is infeasible on the downlink; if the AP encrypts the MAC header, each client will need to decrypt all received packets or use a system like *SlyFi* [13], regardless of whether the packet is meant for it; this could cause high energy drain.

Interference: We assume that interference effects are handled by (a) use of multiple channels (b) carrier sensing or scheduling. Thus, in any contention domain, we assume that only one MIMO transmitter is active at any given time. This is true of current deployments of 802.11n or 802.11ac.

Impact of mobility: In dynamic (mobile) settings, the periodicity of CSI exchanges will be high. This problem however, is inherent to MU-MIMO communications and is not a limitation of *FOG*.

V. Attacker Model

We assume a passive eavesdropper equipped with MIMO. As the number of antennas at the eavesdropper increases, it is more likely that it will successfully decode information. However, in practice, since the distance between any two antennas has to be of the order of the wavelength used for transmission [6], the form factor of an eavesdropper's device will increase with the number of antennas. To reflect a practical scenario (and due to limitations with our testbed), in our evaluations, we limit the number of antennas at the eavesdropper to be no more than the number of antennas at the transmitter. However, *FOG* will still be effective with a reasonable increase in the number of antennas at the eavesdropper, beyond this. One can cope with a more potent eavesdropper by increasing the number of blinding streams sent with legitimate streams.

We assume that the eavesdropper's mobility is not fast enough for its position to change within one packet transmission time. The assumption typically holds true since this transmission time in 802.11ac is of the order of μs.

The eavesdropper knows the protocol in use (802.11n/ac). With MIMO, it can use selective diversity [6] to reduce its BER[1]. It cannot decode the CSI packets since these are encrypted. When the payload is encrypted, its goal is to decode the header (TCP, IP and MAC) bits. If link layer encryption is used, then its goal is to determine the MAC addresses in the transmitted packets. We assume that the eavesdropper will use the information obtained to either perform traffic analysis, spoof MAC or IP addresses etc., and do not explicitly address its role after these bits are obtained.

The eavesdropper can also try to carry out more sophisticated attacks such as those described in [17] and [4]. It knows some transmitted bits (e.g., the AP MAC address) and can then apply a standard filter to try to determine the blinding signals. The attacker then uses an iteratively adjustable filter with this knowledge, to extract other information that cannot be accessed by filtering out the blinding signal. Finally, we

[1]Our experiments with two signal combining techniques, viz., equal gain combining [6] and selective diversity [6] showed that the latter did better.

assume that the attacker does not launch attacks to disrupt the transmissions (e.g., jamming attacks).

VI. Obfuscation with FOG

In this section, we describe the modules of *FOG*. *FOG* takes advantage of encryption at either the transport, network and/or link layers and only tries to blind information relating to exposed headers. This in turn drastically reduces the throughput penalty that is associated with full packet blinding.

Overview: Header blinding can drastically reduce the throughput penalties associated with full packet blinding (e.g., [3]). However, realizing header blinding in practice comes with challenges. We first provide an overview of these below and briefly discuss how to overcome them.

At the PHY layer the header size is not known a priori. Only known is the packet size in bits. The bits from the packet are first scrambled to eliminate long binary strings of zeros or ones [6]. Then, Forward Error Correction (FEC) is applied to generate *coded bits*. To be robust to bursty errors, the coded bits are interleaved. The resulting bit stream is then mapped onto symbols (modulation) and transmitted using OFDM (with 802.11). As evident, the header bits (after the above processes) get mixed intricately with the payload bits. Thus, we need to determine the symbols that correspond to the header bits to perform effective header blinding.

In order to transmit A blinding signals during a header transmission, A *legitimate streams* (out of the N possible streams) will have to be temporarily suspended. In *FOG*, we *switch* across the streams that are suspended during the transmission of different sets of header bits (details later). In other words, from out of the possible N streams, different combinations of A streams are chosen (could be chosen randomly or in accordance to some policy) to be suspended when A blinding signals are to be transmitted. Correspondingly, the blinding signals temporally change and we refer to this process simply as switching (for ease of discussion). Switching significantly helps in cases where the eavesdropper is at a location where a specific blinding signal is not effective (a random sequence of different blinding signals is now used to blind out a packet's header). Note that with this process, temporary suspensions of legitimate streams will need to be indicated to the receivers for the purposes of synchronization.

In a nutshell, *FOG* contains (a) a *header tracking* process that first estimates the header size and then keeps track of the header bits at each stage of PHY layer processing prior to blinding and, (b) a *scheduling* functional process to switch between the suspensions of legitimate streams (and invoking appropriate blinding signals) and indicating these schedules to the receivers. Fig. 1a depicts these processes and how they can be integrated into the 802.11ac architecture. In the following subsections, we describe these processes in detail.

Header tracking: Header tracking consists of four steps; (1) header estimation, and tracking as the bits traverse the (2) scrambler (3) the convolution code (e.g., binary convolutional codes or BCC) encoder and (4) the interleaver.

Header estimation: First, *FOG* needs to estimate the header size (the transport, network and MAC header bits combined). Without interactions between the PHY and upper layers, this is non-trivial. It requires packet inspection at the PHY layer which is complicated and violates the layering. Thus, we conduct an experimental study to estimate the average packet header size for different application layer protocol packets (e.g., HTTP(S), FTP). We collect traces using WireShark [18] at various locations of our campus (again, we were told that no IRB was needed) and examine $\approx$ 150,000 packets. We find that the average size of the header is about 54 bytes in total. We have some details that could allow for more precise estimations (e.g., variance; details omitted due to space constraints), but conservatively, we suggest using a header size of 70 bytes. This is higher than the header size in all the packets from our measurement study and results in a slightly higher overhead but ensures that all header bits are blinded.

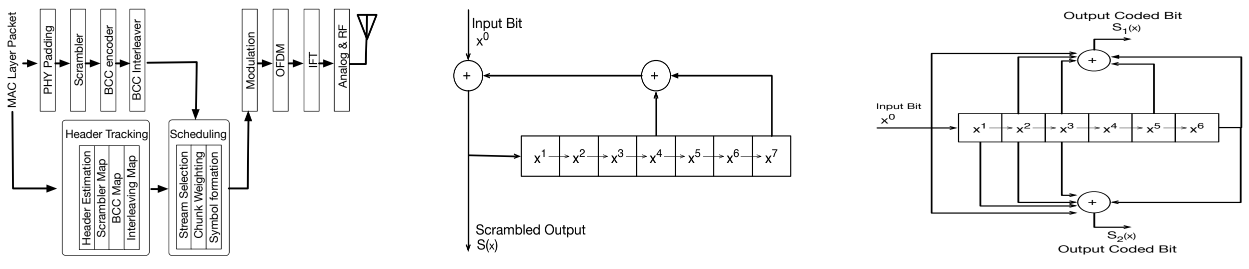
Next, we describe how we track the header during scrambling, FEC and interleaving. We illustrate how the header bits are tracked during the scrambling process. The process for encoding (e.g., using BCC) is similar. We do not describe the descrambling or decoding processes since header tracking is irrelevant to the receiver (more details are found in [6]).

Scrambler Mapping: The scrambler randomizes the bits in a packet, in order to decrease the probability that long binary strings of 1s or 0s exist [6]. The initial seed of the scrambler is 7 bits long and is included in the PHY layer header (and is thus known to the receiver). The function used for scrambling is also known to the both the transmitter and the receiver. The receiver uses the seed value to descramble and retrieve the original sequence of bits. To track the header bits, we taint the output bits of the scrambler if they are header related bits (note that prior to being input to the scrambler, the header bits are at the beginning of the packet). We create an array of indices that record the locations of the header bits, after scrambling. For instance, if there are L header bits, we create an array of length L. Each element of this array points to the location of a unique header bit in the scrambled packet. The memory overhead for maintaining this array for an average size header (54 bytes, i.e. $L = 54 \times 8 =$ 432 bits) is equal to 432 $\times$ the size of an integer, which is around 1.6 KB.

To illustrate, consider an example with 802.11ac [19]. Here, scramble operations are implemented using shift registers as shown in Fig. 1b. There is a generator polynomial $S(x) = x^7 + x^4 + x^0$, known to both the transmitter and receiver. In Fig. 1b, x^1, x^2, ..., x^7 refer to the contents of the register. x^0 corresponds to a bit from the input stream. The initial shift register contents are specified by a 7-bit scrambler seed in physical layer header [19]. Then, the sum bit $S(x) = x^7 + x^4 + x^0$, where the "+" symbol indicates an XOR operation. This sum bit is the first bit of the output sequence. The contents of the shift register are now shifted up by a stage as follows: $x^6 \to x^7$, $x^5 \to x^6$, ..., $x^1 \to x^2$. The new sum bit $S(x)$ is placed in the shift register in place of x^1. The procedure is repeated with the next bit of the input stream (a new x^0).

To taint the header bits, we mark all the output bits that are generated by using even a single header bit. Thus, for each bit of a L bit header, we *mark* (taint) a sequence of $L + 7$ bits in the output sequence since the output sum bit $S(x)$ depends on the seven previous bits.

BCC Mapping: The next step is to track the header bits after FEC. We assume the use of BCC for ease of discussion (tracking with other FEC codes will be similar). In 802.11ac, BCC also uses the shift registers to generate the output bits (Fig. 1c). These registers are initialized with zeros. Each input bit (x^0) generates two output bits using two generator polynomials. For example for a rate $1/2$ code, the generator polynomials are $S_1(x) = x^6 + x^5 + x^3 + x^2 + x^0$ and $S_2(x) = x^6 + x^3 + x^2 + x^1 + x^0$. The scrambler output is



(a) (b) (c)

Fig. 1: (a) Modified 802.11n/ac Architecture. (b) Data scrambler in 802.11ac (c) Convolutional encoder with code rate =1/2

the input to the encoder. These polynomials operate on the contents of the shift register (higher order bits) and a single input bit (the lowest order bit). For illustration, let us assume that the first input bit is 1. Then, two output bits are generated. The first polynomial yields $S_1(x) = x^6+x^5+x^3+x^2+x^0 = 1$ and the second yields $S_2(x) = x^6+x^3+x^2+x^1+x^0 = 1$. In other words, for the first input (scrambled) bit, 1, we get two output coded bits 11. After first bit is processed it is inserted into the shift register and the original contents are shifted; thus, the new contents are 000001. The outputs corresponding to the header bits which were tainted during scrambler mapping are identified and tainted. The tainted output coded bits, include those for which the original header bits are either inputs or part of the shift register.

Interleaver Mapping: The interleaver shuffles the input bits to distribute errors uniformly across the packet; bursty errors are thus eliminated. It consists of an array of R rows and C columns. The input fills the array one row at a time and the output is a read out of the array, one column at a time. The parameters R and C are known to the transmitter, the receiver *and* the eavesdropper. Since the input header bits are tainted, it is easy to track them at the output of the interleaver.

Scheduling: As discussed, when blinding the header bits in a specific stream, some of the other legitimate streams will need to be suspended ; the scheduling process determines these suspensions. For ease of discussion, assume that the traffic is saturated (if not additional blinding streams can be transmitted to improve the level of obfuscation). Given this, the transmitter (with N antennas) seeks to simultaneously send N packets to receivers. Towards realizing the scheduling process, the output bits of the interleaver (for each packet) are divided into chunks of bits. Each chunk, c, has a size of L_c bits. Now there are N streams of chunks, where a chunk may or may not contain the header bits. If a chunk on any of the streams contains a header bit(s), then we choose to suspend one or more of the other streams (depending on the level of protection needed) and instead transmit signals associated with the blinding streams. After the streams (both legitimate and blinding) are chosen, the transceiver system generates the ZFBF weights as described in Section II. The data chunks to be transmitted (including those for blinding) are weighted by the ZFBF beamformer and then OFDM symbols are formed by combining the weighted chunks. In the following, we describe the above steps in detail.

Step 1: Stream Selection: The stream selection step determines on a *per slot* basis, which stream(s) are to be suspended in order to instead send blinding streams (each slot carries a chunk per stream). A queue per stream is created and the indices of all the chunks associated with that stream, that are to be blinded, are recorded. As per a chosen policy, a stream is selected for the subsequent suspension, and a blinding stream is instead generated and sent. In our implementation, we select the stream for suspension randomly as per a uniform distribution. In the long term, this policy ensures fairness between stream suspensions. Given the relatively small number of header bits compared to the packet length, the delays incurred due to stream suspensions and the consequential postponing of chunk transmissions is very small.

To allow the receiver to determine which slots have blinded transmissions, the streams are indexed and the indices of streams are transmitted to the receiver, using ZFBF, prior to the actual data transmission. With knowledge of the suspsensions, the receiver can easily reconstruct the packet from the data received. Transmitting information with regards to the suspended streams constitutes a significant part of the (small) overhead with our approach (this information is represented using a bitmap as described in Section VII).

Fig. 2 depicts an example for how blinding signals are scheduled along with chunks that contain header related bits for a transmitter with $N = 3$, and $A = 1$. In the example, as shown in Fig. 2 (a) three chunks with header related bits are to be scheduled on stream one, and two chunks with header bits are to be scheduled on streams two and three. As shown in Fig. 2 (b) the first stream is suspended first; its chunk is postponed and the bits from a blinding stream are instead transmitted (shaded chunk). All the following data chunks for S1 are delayed by one slot as a consequence. The process is easy to follow in sub figures (c), (d) and (e).

Choice of chunk size: The chunk size defines the required precision for blinding decisions. If $L_c = 1$, then a decision on whether to blind or not is made with respect to each bit. This results in the minimum blinding time and overhead (only the chosen header bits are blinded). However, the legitimate receivers will need to know of suspensions at the "bit level". In addition, switching across modes (legitimate transmission versus blinding) and the computation of ZFBF weights will need to be done every bit and this can be prohibitive. As the chunk size increases, the granularity of blinding becomes coarser (more non-header bits are blinded) and thus, the throughput penalty increases. However, the complexity of the information made known to receivers (which chunks are suspended) and the switching overhead decreases (due to fewer chunks per packet). In the other extreme case, when the chunk size is equal to the packet size, the scheme reduces to *full packet blinding*. The switching overhead is minimized, and there is no need to inform receivers about any suspensions; however, the throughput penalty is the highest.

Step 2: Chunk Weighting: Weighting is part of the blinding process as described in Section II. The weight matrix, W is recomputed each time the AP (transmitter) receives CSI feedback from the receivers. For a N antenna transmitter, there can be a number of different combinations of data and blinding streams according to which transmissions could be performed

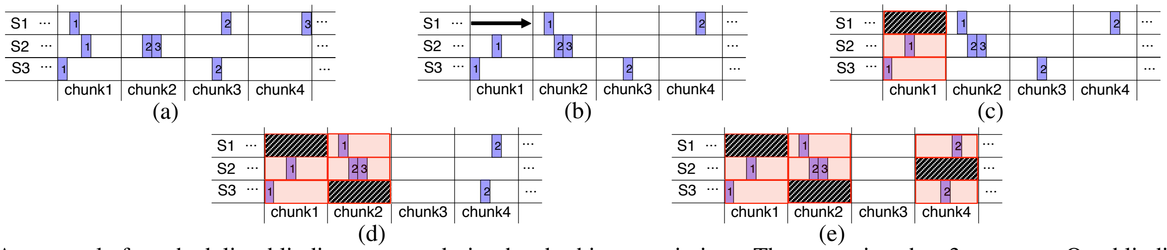


Fig. 2: An example for scheduling blinding streams during header bit transmissions. The transmitter has 3 antennas. One blinding stream is used. The scheduling sequence is from (a) to (e). (a) shows the original layout of the chunks with the header bits. In the first slot stream S1 is chosen for suspension and thus, the transmission of its chunks are postponed by a slot (see (b)); instead a blinding signal is sent (see (c)). In the next slot (see (d)), stream S3 is chosen for suspension, the corresponding chunks are postponed and a blinding signal is transmitted instead. Finally, stream S2 is chosen for suspension (see (e)).

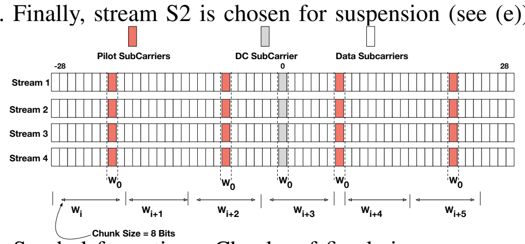


Fig. 3: Symbol formation : Chunks of fixed sizes are mapped onto modulation symbols (e.g., BPSK), weighted differently, and then mapped onto OFDM subcarriers.

(as described in Step 1). For each chunk, W depends on the selected data streams and the blinding streams.

The number of different weight matrices to be computed prior to a packet transmission depends on the total number of streams (N) and the number of desired blinding streams (A). For example, with $N = 3$ and $A = 1$, four weight matrices are needed. These are (i) W_0: no streams used for blinding; (ii) W_1: stream one is replaced with a blinding stream (iii) W_2: stream two is replaced with a blinding stream and (iv) W_3: stream three is replaced with a blinding stream. Each data chunk is weighted based on the *blinding decision*. For the example in Fig. 2, chunks from streams 2 and 3 are being transmitted in the first slot; stream 1 is replaced with a blinding stream. Thus, the weight matrix W_1 is applied to the data during this slot. Similarly, it is easy to see that W_3 will be the weight matrix that is to be used for the chunks to be sent in the subsequent slot. Since different weight matrices are used for different parts of the packet, it becomes much more difficult for an eavesdropper to (a) recover the entire packet at a static location and (b) launch the so called "plaintext attacks against PHY layer security," wherein an attacker tries to recover a single fixed blinding signal that is used for the entire packet duration using known plaintext [4] (discussed later).

Step 3: Symbol formation: After the interleaving stage, multiple data chunks are combined to form OFDM symbols. This symbol formation is based on the 802.11ac standard. We provide a brief description here of how the division of each packet into chunks and the application of different weights for each chunk period, influences this process. We assume that the total number of OFDM subcarriers available is equal to 56 and only 52 of those subcarriers are used for data (as in 802.11ac [14]). Four subcarriers are used for pilot tones; these are typically used to correct frequency offsets, synchronization etc. The rest are used for guard carriers.

The mapping of the chunks to OFDM symbol data is shown in Fig. 3. The bits from each chunk scheduled in a slot, are mapped onto symbols in a modulation constellation (e.g., BPSK has two symbols, QPSK has four). The appropriate weights are then applied on the modulation symbols. To illustrate, let us consider the example shown in Fig. 3, where the chunk size is equal to 8 bits and BPSK modulation is in use. With this modulation, each subcarrier carries one bit. Then, an OFDM symbol (52 subcarriers) can accommodate 6 chunks. In this example, we have 4 data streams and for each data steam the modulation symbols are determined and mapped on to the subcarriers. For each slot (or the chunks scheduled in that slot) the appropriate weight matrix W is applied. In the figure, we see that for each slot, a different weight matrix is used and thus, the set of signals transmitted are different (different desired streams are suspended).

For some chunk sizes, it may not be possible to fill an OFDM symbol with an integral number of chunks; in that case we use filler bits. In our case, the chunk size is equal to 8 bits; then with BPSK, the remaining 4 bits (from the 52 bits) are called filler bits and are randomly generated. The filler bits are discarded after decoding at the receiver.

After mapping, the weighted and modulated symbols from each stream are converted to time domain symbols via FFT and the resulting symbols are transmitted jointly.

Effects at receivers and eavesdroppers: Since the blinding streams are orthogonal to the legitimate receivers' streams, these receivers are unaffected. The received symbols are first mapped onto coded bits and deinterleaved. The decoding of the coded bits is done using the Viterbi algorithm [20], which progressively derive the sequence of raw bits given the coded bits (details can be found in [20]). While the Viterbi algorithm is very effective in correcting isolated errors, it cannot correct long bursty errors. The blinding signals essentially construct a burst of errors at the eavesdropper i.e., the input to the Viterbi decoder first consists of the long sequence of *corrupted* header bits; thus, the decoding will fail.

One could ask if it is possible to reverse engineer the decoding and scrambling by knowing the shift register structures (recall header tracking) and retrieve the header bits. To see why this is impossible let us examine what happens with the shift register corresponding to the scrambler. Similar arguments hold with regards to the BCC encoder, but they are a bit more involved due to the more complex structure of the shift register; we omit a discussion due to space limitations. Assume that initially the scrambler holds the seed values S_1 to S_7 and that these are known to the eavesdropper. The input bit sequence is denoted by I_1, I_2 and so on. It is easy to verify that the first four output bits are $O_i = S_{7-i+1} + S_{4-i+1} + I_i$ for $i \leq 4$. Since these O_i values are either erased or corrupted, the attacker has to guess them. This means that he has to guess the values of I_i and for each of these there are two possible values.

For $4 < i < 8$ it is easy to verify that $O_i = S_{7-i+1} + O_{i-4} + I_i$. Since these are again blinded, and because the eavesdropper does not know O_{i-4}, he has to again guess the values of I_i. One can follow through the expressions for $i \geq 8$, and it is easy to see that the attacker can at best only guess the value of a blinded bit. Thus, the number of possible combinations that have to be considered to infer the header bits will be $O(2^H)$. If $H = 400$ bits, this corresponds to $\approx 2^{400}$ combinations!

Robustness to plaintext attacks against PHY layer security: Plaintext attacks against PHY layer security [4] try to reduce the effectiveness of a blinding signal. The attack depends on some part of the header being known. The attacker constructs a filter that estimates the weighting matrix W. Traditionally since a fixed blinding stream is generated, an attacker can slowly adjust the coefficients of a filter to estimate W and decode with less than a 10% bit error rate [4].

Unlike previous schemes, *FOG* varies the blinding signal depending on which stream is suspended as discussed earlier. Let the transmitter possess N antennas. For simplicity, assume that only one data stream is suspended and a blinding stream inserted in its place. The total number of weights used will then be $N+1$ (N possibilities for the cases where each specific stream is suspended and one for the case where none are suspended). Assume that the attacker knows that the header bits are being blinded and only one blinding stream is used. However, for each header bit, it is unware of which weighting matrix W_j is used, where $\jmath \in \{1, 2, \ldots, N\}$ (assuming that W_0 is used in the case without blinding). If the attacker is able to determine the weights using a known field (e.g., the AP's MAC address), it will still have to consider all possible combinations of these weights for the remainder of the header. If there are H such header bits, each bit can be encoded with *any* of the N weights, and thus, there are N^H possibilities to consider. For a 50 byte header, if the transmitter has 4 antennas, this corresponds to 4^{400} possibilities (exponentially large). The process becomes more difficult if the attacker is unaware of how many streams are used for blinding. For example, if the transmitter hops randomly between 1 and 2 blinding streams, the likelihood of decoding will further decrease.

VII. Testbed and Implementation

Testbed: Our experiments are on our WARPlab testbed [5] with Versions 1 and 3 (V1 and V3) WARP boards. We implement the transmitter and the eavesdropper on the V1 boards since they can be used for MIMO communications with four antennas. We use the V3 boards for the receiver nodes (single antenna). The transceivers were at fixed locations but the eavesdropper was moved to different locations.

Topology: The default topology consists of one transmitter, four receivers and one eavesdropper. We form a 6×5 grid (60 in. $\times$ 50 in.); the transmitter is placed in the middle of the grid and the receivers at the corners. The eavesdropper is placed at 25 different locations at grid intersections. The packet sizes we use, are 240 and 1514 bytes unless otherwise stated. Note that the maximum distance between the transmitter and the receivers is governed by the hardware/power limitations of WARP boards. With higher powers, larger distances can be covered and our results will apply.

Implementation: The blinding schemes are implemented as a thin sublayer within the physical layer. Our implementation consists of three modules; (i) the header tracking module, (ii) the scheduling module and (iii) the communication module. The first two modules are described in Section VI. The communication module (COMM) is responsible for exchanging the CSI information between the transmitter and receivers and for performing transmissions. We implement the explicit feedback scheme used for beamforming as described in the 802.11ac standard [19]. First, the transmitter periodically broadcasts a request for the CSI. Then, the COMM module at each receiver encrypts and transmits the CSI; the transmitter and each receiver share an a priori loaded secret key and use AES for encryption [21]. At the transmitter, the CSIs are received and decrypted by COMM. The H and W matrices are constructed based on this CSI.

We also implement the scrambler, the BCC encoder and interleaver from the IEEE 802.11ac specification [19]. After scheduling, COMM constructs the PHY frame. A bitmap specifying which transmissions are suspended during the blinding process, is included after the service bits (Scrambler Seed, Reserved bit and CRC), in the frame. The bitmap represents each chunk in the packet by a bit; an entry of '0' indicates a suspension, and '1,' otherwise. The bitmap is also encrypted using AES. Note that the Cyclic Redundancy Check (CRC) is calculated over the modified payload which contains this bitmap. If the bitmap is corrupted, the packet is retransmitted.

VIII. Evaluations

We extensively evaluate *FOG* experimentally. For comparison we also implement a baseline case where the entire packet is blinded (full packet blinding) with a single fixed blinding signal as in [3].

Notation: We refer to blinding the entire packet as *PB* (for Full "Packet Blinding"). If A blinding streams are used we refer to it by PBA; e.g., $PB2$ denotes full packet blinding with two blinding streams ($A = 2$). Similarly the notation FA refers to *FOG* with A blinding streams used for header protection (e.g., $F2$ refers to the case where two blinding streams are used to protect header bits). The case without blinding is denoted by ZFBF.

The effect of blinding on the eavesdropper: First we perform experiments to determine the effectiveness of blinding on eavesdropping. We use PB; *FOG* is not used. $N = 4$ and A is varied. When $A = 0$, no blinding streams are transmitted i.e., four legitimate data streams are sent. Fig. 4 plots the percentage of positions where the eavesdropper (with 4 antennas) perceives a bit error rate (BER) of less than or equal to 0.1, with respect to decoding one or more data streams. Here (and henceforth), we assume conservatively, that an eavesdropper can somehow recover (either by using additional antennas compared to what is shown in the results or by a brute force search) a packet with this BER (of 0.1); in reality, a much more stringent requirement on BER (lower) will be necessary for successful recovery.

Without blinding, the eavesdropper can decode at least one data stream in approximately 72% of the locations. This percentage goes down to 5% when three blinding streams are used. This dramatic improvement is due to the increase in power allocated to the blinding streams, which significantly hurts the eavesdropper in decoding the legitimate data. Fig. 5 shows the fraction of locations where the eavesdropper can decode with a probability lower than a certain threshold (specified on the x-axis) in the absence of any blinding streams. We see that the probability of the eavesdropper decoding a data stream, increases significantly if the eavesdropper uses

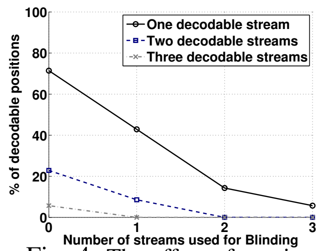


Fig. 4: The effect of varying the number of streams used for blinding on security.

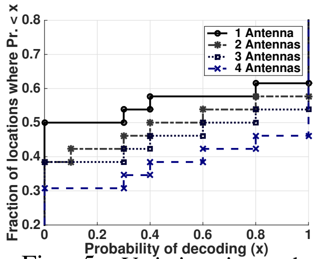


Fig. 5: Variation in prob. of decoding by eavesdropper equipped with one or more antennas.

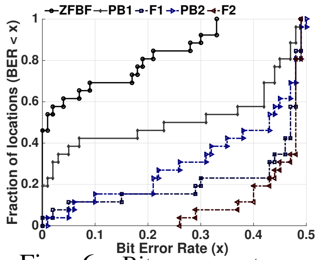


Fig. 6: Bit error rate as perceived by the eavesdropper for different blinding schemes.

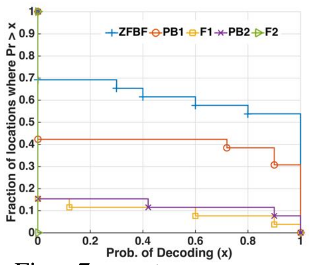


Fig. 7: Packet error rate as precevied by the eavesdropper for different blinding schemes.

Version	Header	After Convolution	Number of blinded bits	% of pkts blinded
F1	432 bits	876 coded bits	936 coded bits	3.86%
F2	432 bits	876 coded bits	952 coded bits	3.93%

TABLE I: Number of bits blinded using *FOG*

multiple antennas for reception. By using four antennas, the percentage of the locations where the eavesdropper can decode more than 80% of the time, increases from 40% to 55%, on average as compared to the case where it has a single antenna. This is because the probability that the received signal is either distorted or destroyed completely on all antennas is much lower than it being so on a single antenna.

Efficacy of *FOG*: Next, we evaluate *FOG* in terms of both bit and packet error rates (BER and PER) seen at the eavesdropper with 4 antennas. Note that if a packet is correctly decoded (CRC passes), only the header bits are exposed (payload is encrypted). Fig. 6 shows the fraction of eavesdropper locations where the BER observed was less than what is shown on the abscissa (CDF). Without any blinding, in 50% of the locations the BER at the eavesdropper is $\leq 1\%$. By using one or two streams for PB, this BER increases to 10% and 30%, respectively, in 40 % of the locations (even with convolutional coding). This is because the decodability at the eavesdropper is hurt by the blinding signal(s).

Interestingly, we observe in Fig. 6 that *FOG* provides better obfuscation than PB. This is due to different blinding signals being used during the transmission of different parts of the header, for obfuscation. Thus, at an eavesdropper location where a single (static) blinding stream is ineffective, this combination of blinding signals provides effective obfuscation (causes higher BERs). In some cases, FOG *with one blinding stream, even outperforms the PB with two streams!* For example, with $F1$,almost all locations considered experience a BER of 40 % or higher; however with $PB2$, the BER experienced is 10 % in approximately 18 % of the considered locations.

In Fig. 7, we plot the fraction of locations where, the ability of the eavesdropper to recover packets (1-PER) is greater than the probability of decodability shown on the x-axis (complementary CDF). We see that without blinding, the probability of recovery is significant in over 30 % of the locations. With $F2$, the probability of recovering packets is 0 % in 100 % of the locations (all the points are at the origin in the figure). With $F1$, the probability of recovery is less than 10 % in over 90 % of the locations. This again, showcases the effectiveness of *FOG*.

Eavesdropper's ability to decode headers: Next, we run a micro benchmark where, the attacker collects all retrievable packets (decodable or corrupted). We compare a sample set of fields in these packets with ground truth. If there is a match, we assume that the attacker has access to that field. The results shown in Fig. 8, show that without blinding, the eavesdropper can retrieve many of the header fields in 30 to 35 % of the considered locations. With blinding, the eavesdropper can only retrieve a particular header field in at most 8% of the locations.

As in earlier results, increasing the number of blinding streams protects better and *FOG* is more effective than PB.

Throughput performance: Next, we quantify the performance of *FOG* in terms of the degradation in the average achieved saturation throughput and compare it with that of PB. Fig. 9 plots the degradation in throughput with one blinding stream. With *FOG*, we use chunk sizes of 8 bits in these experiments. As one might expect, there is a drastic reduction in the throughput degradation with *FOG*. $PB1$ essentially decreases the number of legitimate streams by 1 (from 4) and thus, the throughput hit is about 25 %. With *FOG*, we see that the throughput hit drops to 5 % or less.

The throughput penalties with *FOG* can be divided into two parts as shown in Fig. 9. Specifically, there is a throughput hit because of blinding of the chunks that contain header bits (and some non-header bits as collateral); this is referred to as *blinding overhead*. In addition, there is the overhead of adding information with regards to when a stream is suspended in order for the transmitter to send a blinding signal. This overhead contributes to an overall throughput degradation (longer packets) and is referred to as *tracking overhead*.

Effect of chunk size on throughput: In Fig. 10, we show the effect of varying the chunk size on the throughput degradation. As discussed above, there is a throughput degradation due to (a) blinding, and (b) the additional overhead of suspension tracking (tracking overhead). As the chunk size in bits decreases, the tracking overhead increases. This is because there are more chunks and one needs to indicate to the receivers which chunks are being suspended. However, the blinding overhead itself decreases. As the chunk size increases we see the opposite effect; the blinding overhead increases since we are blinding larger chunks (and thus, more unnecessary bits) but the tracking overhead decreases. A chunk size of 8 bits seems to be the optimal and that is what we are using as the default setting in our implementation.

Table I captures the number of "extra" bits that are blinded with *FOG* with a header of size 432 bits (packet size is 1514 bytes). After convolution coding the number of bits get doubled (coded bits $\approx$ twice the original bits). The reported results only consider the extra bits that are blinded from one data stream (other streams may also be blinded as a consequence of this, but we consider that as collateral i.e., an additional benefit). We see that the overall "fraction" of the packet that is blinded is less than (3.86% in the F1 case) with *FOG*. In the case of two blinding streams, a slight increase

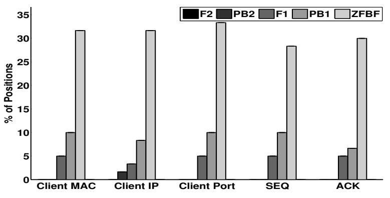


Fig. 8: Probability of decoding of specific header fields with different blinding schemes.

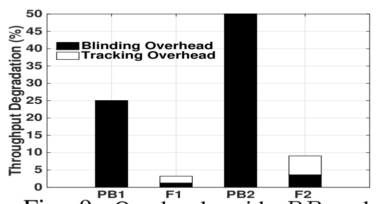


Fig. 9: Overheads with PB and *FOG*.

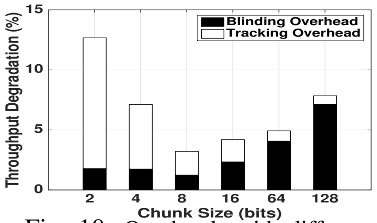


Fig. 10: Overheads with different chunk sizes in *FOG* (one blinding stream is used).

in the number of coded bits is observed. This is because, with two blinding streams, the likelihood of a stream being suspended increases. This reduces the possiblity that chunks from different streams have common header bits (recall Fig. 2). Thus, there is a slight increase in the number of slots that carry blinding signals and thus, a slight increase in overhead.

IX. Related Work

Physical layer blinding attacks: In [17] and [4], a PHY layer passive attack that resembles known plaintext attacks in cryptography is identified. The attacker guesses part of the data transmitted by a sender (e.g., MAC header bits). Then, iteratively, it filters the received data and compares the known parts of the original data with the derived data (filter output) until the difference between the filter output and the expected original data is minimized. The work in [22] improves this attack capability via techniques to minimize the amount of the data required for the attack to be successful. However, the proposed attacks implicitly assume that the header information is transmitted at the beginning of the packet and known to the attacker. However, 802.11n/ac's scrambling and interleaving processes distribute the header bits across the entire packet. More importantly, these attacks implicitly assume that a single blinding signal is used by the transmitter (as in traditional full packet blinding). Since a combination of blinding signals are used in *FOG*, these attacks are much harder if not impossible.

Friendly Jamming: With friendly jamming (e.g., [23], [24]), the idea is to generate jamming signals and control the decodability of these signals by sharing secret keys between the legitimate transmitter and receiver. The jamming signal is not decodable by any other receiver (eavesdropper) without the key. Friendly jamming, typically carried out by other nodes, increases the interference and thus, could reduce the overall network capacity. Furthermore, it requires coordination across nodes. MIMO based blinding does not have these issues.

X. Conclusions

In this paper, we design and implement *FOG*, a practical framework for obfuscation of packet headers transmitted in the open in wireless streams, using MIMO. *FOG* tracks header bits as they traverse an intricate PHY layer consisting of scrambling, application of FEC and interleaving. *FOG* provides protection against attacks that try to decipher a fixed blinding signal (as traditionally used). We show that *FOG* provides enhanced obfuscation of wireless streams compared to state of the art approaches that use MIMO to blind out entire streams; at the same time, it consumes much lower overhead.

Acknowledgement

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. The work is also supported by National Science Foundation under Grant #1464410.

References

[1] C. Shuo, W. Rui, W. Xiaofeng, and Z. Kehuan, "Side-channel leaks in web applications: a reality today, a challenge tomorrow," in *IEEE Symposium on Security and Privacy*, 2010.

[2] A. Proano and L. Lazos, "Selective jamming attacks in wireless networks," in *IEEE ICC*, 2010.

[3] N. Anand, S.-J. Lee, and E. Knightly, "Strobe: Actively securing wireless communications using zero-forcing beamforming," in *INFOCOM*, 2012.

[4] M. Schulz, A. Loch, and M. Hollick, "Practical known-plaintext attacks against physical layer security in wireless MIMO systems," in *NDSS*, 2014.

[5] "Rice University WARP Project," http://warp.rice.edu.

[6] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. NY, USA: Cambridge University Press, 2005.

[7] T. Yoo and A. Goldsmith, "On the optimality of multiantenna broadcast scheduling using zero-forcing beamforming," *IEEE JSAC*, 2006.

[8] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall, "802.11 user fingerprinting," in *MobiCom*, 2007.

[9] Q. Sun, D. Simon, Y.-M. Wang, W. Russell, V. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *IEEE Symposium on Security and Privacy*, 2002.

[10] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted VoIP traffic: Alejandra y roberto or alice and bob?" in *USENIX Security Symposium*, 2007.

[11] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *J. Mach. Learn. Res.*, 2006.

[12] M. Junaid, M. Mufti, and M. U. Ilyas, "Vulnerabilities of IEEE 802.11i Wireless LAN CCMP Protocol," *Enformatika*, vol. 11, p. 228, Feb 2006.

[13] B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall, "Improving wireless privacy with an identifier-free link layer protocol," in *ACM MobiSys*, 2008.

[14] M. Gast, *802.11Ac: A Survival Guide*. O'Reilly Media, Inc., 2013.

[15] C. V. Wright, F. Monrose, and G. M. Masson, "802.11w - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 4: Protected management frames," *IEEE Standard for Information technology*, 2009.

[16] "Iphone 6 specification," https://www.apple.com/iphone-6/.

[17] N. O. Tippenhauer, L. Malisa, A. Ranganathan, and S. Capkun, "On limitations of friendly jamming for confidentiality," in *IEEE Symposium on Secuirty and Privacy*, 2013.

[18] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce, *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[19] E. Perahia and R. Stacey, *Next Generation Wireless LANs: 802.11 n and 802.11 ac*. Cambridge university press, 2013.

[20] J. Forney, G.D., "The Viterbi algorithm," *Proc. of the IEEE*, 1973.

[21] "Advanced Encryption Standard (AES) (RFC 3826)," http://www.ietf.org/rfc/rfc3826.txt.

[22] Y. Zheng, M. Schulz, W. Lou, T. Hou, and M. Hollick, "Highly efficient known-plaintext attacks against orthogonal blinding based physical layer security," *IEEE Wireless Communications Letters*, 2015.

[23] W. Shen, P. Ning, X. He, and H. Dai, "Ally friendly jamming: How to jam your enemy and maintain your own wireless connectivity at the same time," in *IEEE Symposium on Security and Privacy*, 2013.

[24] H. Rahbari and M. Krunz, "Friendly cryptojam: A mechanism for securing physical-layer attributes," in *ACM WiSec*, 2014.