Shadow Wi-Fi: Teaching Smartphones to Transmit Raw Signals and to Extract Channel State Information to Implement Practical Covert Channels over Wi-Fi

Matthias Schulz
Secure Mobile Networking Lab
TU Darmstadt, Germany
mschulz@seemoo.de

Jakob Link
Secure Mobile Networking Lab
TU Darmstadt, Germany
jlink@seemoo.de

Francesco Gringoli
CNIT
University of Brescia, Italy
francesco.gringoli@unibs.it

Matthias Hollick
Secure Mobile Networking Lab
TU Darmstadt, Germany
mhollick@seemoo.de

ABSTRACT

Wi-Fi chips offer vast capabilities, which are not accessible through the manufacturers' official firmwares. Unleashing those capabilities can enable innovative applications on off-the-shelf devices. In this work, we demonstrate how to transmit raw IQ samples from a large buffer on Wi-Fi chips. We further show how to extract channel state information (CSI) on a per frame basis. As a proof-of-concept application, we build a covert channel on top of Wi-Fi to stealthily exchange information between two devices by prefiltering Wi-Fi frames prior to transmission. On the receiver side, the CSI is used to extract the embedded information. By means of experimentation, we show that regular Wi-Fi clients can still demodulate the underlying Wi-Fi frames. Our results show that covert channels on the physical layer are practical and run on off-the-shelf smartphones. By making available our raw signal transmitter, the CSI extractor, and the covert channel application to the research community, we ensure reproducibility and offer a platform for further innovative applications on Wi-Fi devices.

1 INTRODUCTION

Wi-Fi can be regarded as the de-facto standard for wireless local area networking, and the installed base is in the billions. Adhering to the Wi-Fi standard provides for interoperability and serves the basic communication needs such as Internet access. Most Wi-Fi chips integrate more advanced features, which are usually neither documented nor exposed to developers or end users. In this work, we use such features to demonstrate how to transmit arbitrary, time-limited waveforms in the 2.4 and 5 GHz bands with up to 80 MHz bandwidth using Wi-Fi chips found in off-the-shelf devices such as smartphones. As a result, we get a software-defined radio (SDR) that is limited by its buffer size for storing samples but that has the benefit of maintaining the device specific transmit and receive characteristics by using the transceiver and antenna setups found in off-the-shelf devices. The latter allows to extend lab experiments to wide-spread commercial end-user platforms. Compared to existing SDR platforms, we enhance mobility by adding SDR functionalities to smartphones and we support low cost devices such as the Raspberry Pi B3+ (roughly USD 45) which allows large-scale experiments with hundreds of nodes at moderate costs.

While SDRs provide for ample flexibility, the overhead for generating raw signals in software can be prohibitive. In contrast, the use of dedicated hardware for signal processing allows to efficiently modulate sequences of bits into wireless signals and vice versa. Especially on the receiver side, continuous calculations are required to perform correlations to detect incoming frames. Hence, for transforming Wi-Fi chips into SDRs, it is beneficial to use as many of the existing dedicated signal processing units as possible for both the sending as well as the receiving path. For the receiving path the following example illustrates this trade off. During regular reception, every Wi-Fi receiver needs to first extract channel state information (CSI) from the long-term training field (LTF) of a frame's preamble to cancel the effects of the wireless channel and to demodulate the transmitted data. If we aim at implementing applications that rely on CSI, they should avoid performing CSI extraction on a sample buffer on their own, but leverage the already existing information instead. As one part of our solution, we, hence, show how to extract CSI on a per-frame basis for use in advanced applications. For the sending path, the goal is to support sending raw IQ samples from a large buffer to enable SDR capabilities. This allows, amongst others, to modify regular Wi-Fi transmissions in arbitrary fashion, which is a deterministic task easily implementable.

As a proof-of-concept application that uses both SDR-like transmissions and CSI extraction capabilities, we chose to implement a new physical layer-based covert channel. By means of this covert channel, it is possible to stealthily embed additional information into Wi-Fi frames. Ideally, this should not impact the reception of such frames by normal receivers. Yet it allows to covertly exchange information between two devices Alice and Bob that can observe each other's radio communications. Similar to most physical layer covert channels, this channel can be detected and decoded by an eavesdropper (Eve), in case she knows the implementation details

MobiSys '18, June 10–15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5720-3/18/06…$15.00
https://doi.org/10.1145/3210240.3210333

of the channel and has access to similar SDR-like functionality or advanced signal analysis capabilities on the physical layer. In contrast, this channel will be indiscernible for unmodified off-the-shelf devices, which do not allow for user access to physical layer parameters. To demonstrate the advanced Wi-Fi capabilities of our solution, we introduce *Shadow Wi-Fi*, a physical layer-based covert channel over Wi-Fi. The covert channel itself works by pre-filtering outgoing Wi-Fi frames and encoding secret information into the filter. We can observe the filters' effects at a receiver by evaluating the per-frame channel state information.

Summarizing, our contributions are as follows:

- We unleash raw signal transmission capabilities on Wi-Fi chips used in off-the-shelf smartphones.
- We extract per-frame channel state information on smartphones and make it available to the user space.
- We implement *Shadow Wi-Fi*, a Wi-Fi covert channel based on prefiltering of outgoing Wi-Fi frames on the physical layer.

To ensure reproducibility of our results, we provide our raw signal transmitter[1], the CSI extractor[2], and the covert channel application[3] to the research community as open source implementation.

The remainder of this paper is structured as follows. We discuss related work in Section 2, followed by an explanation of the background on Wi-Fi chip internals in Section 3. The design of our covered channel is introduced in Section 4, followed by implementation details of our solution detailing all subsystems in Section 5. We perform a thorough evaluation of our system with particular emphasis on the covert channel performance in Section 6. We discuss the obtained results in Section 7 and conclude in Section 9.

2 RELATED WORK

We cluster our related work into the four domains: (1) software-defined radios, (2) channel state information extraction, (3) covert channels, and (4) cross-technology communication.

2.1 Software-defined radios

SDRs implement radio system components in software that would typically be implemented in hardware on dedicated devices. Only antennas with radio frequency front end and analog-to-digital or digital-to-analog converters remain as indispensable hardware components. All protocol dependent signal processing steps are handled by programmable hardware or software. The Universal Software Radio Peripheral (USRP) is a product family of well-known SDRs designed by Ettus Research [5]. Its products operate from DC to 6 GHz, including multiple-input multiple-output (MIMO) systems. Low cost variants provide a bandwidth of 56 MHz. HackRF One is a comparable SDR peripheral from Great Scott Gadgets which provides a frequency range from 1 MHz to 6 GHz, but is limited to one antenna and 20 MHz bandwidth [10]. Lime Microsystems offers the LimeSDR—a MIMO-capable SDR with a bandwidth of 61.44 MHz and frequency range from 100 kHz to 3.8 GHz [18]. Mango Communications offers the WARPv3 hardware [3], a wireless platform operating in the 2.4 and 5 GHz bands with a bandwidth of 40 MHz. Our smartphone SDR is as well limited to those two frequency bands, but offers a comparably high bandwidth of up to 80 MHz. While the above mentioned SDR platforms can continuously transmit and receive signals and even process them in real time on a field programmable gate array (FPGA), our implementation is (currently) limited to transmitting samples from a space limited buffer. This mode of operation is similar to the WARPLab Reference Design for WARPv3 SDRs that allows to generate signals in MATLAB, load them into the WARP's internal buffers and trigger the transmission of the buffer contents. A better understanding of the Wi-Fi chips' direct memory access (DMA) controllers may also allow to refill our transmit buffers in time to allow continuous transmissions with our platform. Another SDR often used for Wi-Fi research is the Microsoft Research Software Radio (Sora) that provides 40 MHz bandwidth [25] and MIMO-capabilities.

2.2 Channel State Information (CSI)

CSI represents the channel properties between sender and receiver on individual subcarriers. It is estimated in OFDM systems in order to compensate channel impairments. However, the CSI is not accessible on off-the-shelf devices or commodity Wi-Fi cards by default. Halperin et al. provide a closed-source CSI extraction tool for Intel Wi-Fi Link 5300 wireless NICs [11]. Xie et al. present an open-source solution for Atheros Wi-Fi NICs [27]. Both are limited to 802.11n operation, while our CSI extractor is open source and has 802.11ac capabilities to extract CSI with up to 80 MHz bandwidth. In practice CSI can be used for various applications. For example, Ricciato et al. use CSI extracted from Broadcom's 802.11g Wi-Fi chips to enhance the resolution of time-of-arrival measurements used in velocity and position estimations [19]. Bagci et al. use CSI to detect whether someone tampered with a device [1]. Jiang et al. use CSI to detect spoofing attacks in Wi-Fi networks [14].

2.3 Covert Channels

So called covert channels use legitimate communication channel properties to transmit additional data that is invisible to uninitiated receivers. The term was first used by Lampson in 1973 [16]. Network based approaches were introduced by Girling and Wolf in 1987 and 1989 [6, 26]. Szczypiorski first addressed Wi-Fi covert channels in 2003 [23]. Szczypiorski et al. present a physical layer covert channel approach that imposes on unused padding bits in physical layer data units of Wi-Fi, WiMAX, and LTE [7, 8, 24]. Hijaz and Frost occupy unused subcarriers inside a protocol's spectrum with covert information [12]. Hudhajanto et al. suggest using pseudo random fake subcarrier locations to hide information [13]. Dutta et al. hide symbols by replacing existing constellation points with additional subconstellation points [4]. Grabski and Szczypiorski replace the cyclic prefix (CP), used to reduce inter symbol interference (ISI), for hiding data [9]. Classen et al. enhance the idea by replacing CPs partially and distributing overlapping covert samples to several CPs [2]. They also introduce phase shifts in short-training field (STF) symbols and introduce artificial carrier frequency offsets (CFOs) into consecutive OFDM symbols to transmit hidden information.

The covert channel that we design in this work is novel. To be practically realized, it requires modifications at the transmitter that affect the CSI measurements at the receiver. Transmit beamforming

[1]Raw signal transmitter's source code: https://nexmon.org/sdr
[2]CSI extractor's source code: https://nexmon.org/csi_extractor
[3]Covert channel's source code: https://nexmon.org/covert_channel

is a technology that allows such modifications by applying filters to outgoing signals. Kumar et al. enable beamforming on commodity Wi-Fi cards [15], but their software is closed source and, hence, cannot be modified for our purposes.

2.4 Cross-Technology Communication

Today's wireless technologies often coexist in common frequency bands. This can be used for cross-technology communication (CTC), where heterogeneous wireless devices communicate directly. However, off-the-shelf devices often do not provide the generation of correct signals for other technologies by default. Li and He demonstrate that Wi-Fi chips can generate Wi-Fi frames whose payloads form correct ZigBee frames that can be used to control ZigBee devices [17]. The SDR-like signal transmission feature we present in this work is even more versatile, as we are not limited to signals that can be imitated with Wi-Fi's physical layer. Instead, we can modulate any frame format in the 2.4 and 5 GHz bands that has a bandwidth of up to 80 MHz and whose frames fully fit into the D11 core's Template RAM. For bi-directional communication, the signal capture feature of Broadcom Wi-Fi chips needs further exploration and one has to consider the delay introduced by reading and writing values from and to the Template RAM.

3 WI-FI CHIP INTERNALS

The Wi-Fi chips considered in this work are all Broadcom or Cypress devices. We illustrate their internals in Figure 1. In this work, we focus on FullMAC devices as they are generally used in smartphones. Compared to SoftMAC chips, FullMAC chips contain an additional ARM processor that implements MAC-layer processing functions in firmware and acts as an Ethernet-to-Wi-Fi bridge to the driver. Their time-critical MAC and PHY implementations equal those of SoftMAC chips so that our findings are generally applicable to both. In Figure 1, we present a high level overview of the Wi-Fi chip's frame handling process including components required for raw signal transmissions and CSI extraction that we present in the following subsections.

3.1 Regular Wi-Fi operation

During regular Wi-Fi operation, the kernel generates Ethernet frames on the host, and sends them through the driver to the Wi-Fi chip's ARM processor. The ARM firmware converts the Ethernet frames into Wi-Fi frames and passes them on to the D11 core that is responsible for real-time MAC processing. To this end, direct memory access (DMA) controllers copy frames into TX FIFOs. Here, another firmware—called ucode—implementing a programmable state machine (PSM) in the D11 core decides how to handle the frame transmission by the physical layer, where frames are modulated and then passed to the analog front end for upconversion to the transmission frequency and amplification. In [21], we describe the architecture of Broadcom Wi-Fi chips in detail and present our Nexmon firmware patching framework used to modify both the ARM firmware and the ucode.

For reception, frames generally traverse the same path in reverse but with additional processing steps. The physical layer first needs to detect the presence of a frame and correlate with the long-training field of the preamble to find the frame's exact starting point.

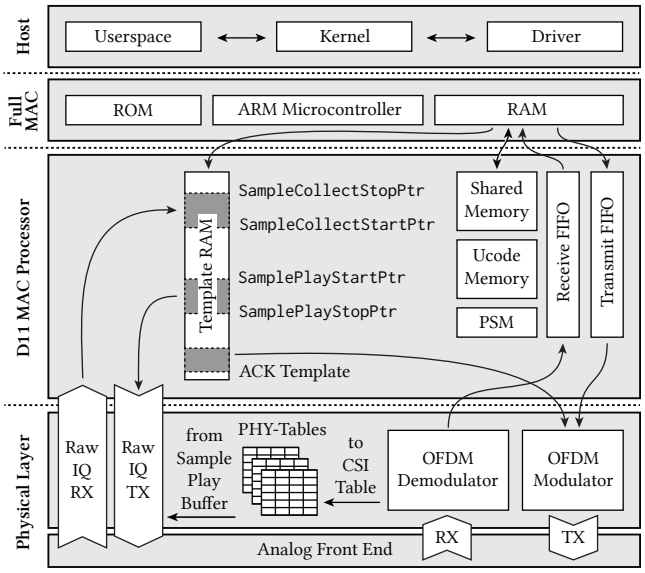

Figure 1: Internals of Broadcom and Cypress FullMAC Wi-Fi chips illustrating regular frame transmit and receive paths as well as raw signal handling capabilities.

This is required to separate the received orthogonal frequency-division multiplexing (OFDM) symbols. They are demodulated by applying a fast Fourier transform (FFT) to extract quadrature amplitude modulated (QAM) symbols for each Wi-Fi subcarrier. Due to fading on the wireless channel, amplitudes and phases change between transmitted and received symbols for each subcarrier. To reverse this effect, the physical layer needs to estimate amplitude and phase changes by dividing the received long-training field symbols by their known transmitted equivalents for each subcarrier. The result is called channel state information (CSI) that is inverted and applied to the received data symbols to extract the transmitted symbols. The CSI is stored in so called physical layer tables, which are memory regions in the physical layer core accessible by a table identifier and an offset. The physical layer demodulates the extracted symbols and stores the resulting bytes in the RX FIFOs where they can be processed in real time by the D11 core's firmware (ucode).

If the ucode detects an incorrectly received frame or one that is not addressed to the receiver's MAC address, the frame will be dropped. Otherwise, the ucode triggers the DMA controller to copy the frame into the RAM of the ARM processor and schedules the transmission of an acknowledgement from Template RAM. Among other things, this memory contains templates for acknowledgement frames and is, for example, 512 KiB in size on a BCM4358 chip. Converting the received frame into an Ethernet frame and sending it to the host system finalizes the frame reception operation.

3.2 Raw Signal Processing

Besides processing regular Wi-Fi frames, we can also operate the Wi-Fi chip as a software-defined radio which is an undocumented feature we found by reverse engineering Wi-Fi firmwares. In [20],

we first described raw signal transmissions on Wi-Fi chips. We used the physical layer sample-play table (see Figure 1) to store IQ samples and trigger a transmission to reactively jam Wi-Fi frames. As the sample-play table only holds up to 512 samples on a BCM4358 chip, it is only useful for transmitting very short or repeated signals—originally intended for calibration purposes. The so called IQ samples describe the inphase (I) and quadrature (Q) components of quadrature modulators used to upconvert a complex baseband signal $I + jQ$ to a carrier frequency f_c (e.g., $f_c = 2.412$ GHz for transmitting on Wi-Fi channel 1) resulting in the radio frequency signal s_{RF} that gets amplified and transmitted by the antenna: $s_{\text{RF}}(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t)$. By gaining the ability to freely modify the values of I and Q, we are able to modulate any signal within a given bandwidth (e.g., 80 MHz) around the carrier frequency.

In this work, we introduce means to use up to the whole Template RAM to store IQ samples for transmission and reception. This memory holds 131072 IQ samples on a BCM4358, which is sufficient for over 3 ms of raw signal (Table 1 gives a capability overview for different chips). We can trigger the capture or transmission of raw signals to and from Template RAM by writing to the D11 core's `psm_phy_hdr_param` (core revision < 50) or `SampleCollectPlayCtrl` (core revision ≥ 50) registers. Using the `SampleCollectStartPtr` and `SampleCollectStopPtr` registers, we can define where received raw samples are stored. The two registers `SamplePlayStartPtr` and `SamplePlayStopPtr`, respectively, define where raw samples are stored for transmission. Using these registers, off-the-shelf smartphones can be turned into SDRs that operate on signals that fit into Template RAM. The sample play registers are only available on D11 cores used in combination with 802.11ac physical layers. Hence, they are not available on devices with other PHYs. We use the SDR transmission features to transmit our covert data containing Wi-Fi frames in our proof-of-concept application described in Section 4.

3.3 Reverse Engineering Process

Our reverse engineering attempts of the raw signal processing features started after discovering the `wlc_phy_tx_tone_acphy` function in wl SoftMAC driver binaries that contain symbols. It is used in calibration functions to generate and play a single tone using the digital-to-analog converters. However, simply calling this function does not result in a tone transmission on the antenna. To enable the tone transmission, we experimented with various configurations and realized that tones get transmitted after setting the transmit amplifiers to manual power control. As transmitting single tones was not sufficient for our purposes, we analyzed the tone transmission function and saw that a CORDIC function generates IQ samples based on a given frequency. Those samples are then written into the physical layer's sample-play table. Then the `wlc_phy_runsamples_acphy` function is invoked to start the signal playback by setting the corresponding physical layer registers. By generating our own IQ samples, we were able to send arbitrary waveforms that fit into the up to 512 samples holding sample-play table. Due to the small size, only very short or periodic signals are implementable. Thanks to a header file, we found out that the last parameter of the runsamples function is called `mac_based`. Though it was never set in the firmware binaries we analyzed, we deduced that there had to be an alternative place to store samples. By further analyzing wl SoftMAC drivers, we found the two functions `wlc_phy_sample_collect_acphy` and `wlc_phy_sample_data_acphy`. The first one triggers the collection of raw samples into the Template RAM and the second one retrieves the collected samples from Templates RAM. Analyzing the sample collect function, we identified that it first checks whether mac-based sample play is used and either allocates only half of or the whole Template RAM for sample collection. Hence, we concluded that mac-based sample play allows to play samples from Template RAM. Additionally, the function pointed us at the D11 core's registers to define the start and end of the sample collect section. Their names are available in the `d11.h` header file. Close to these register names, we found those defining start and end pointers for sample play. Simply loading IQ samples into the Template RAM, setting the pointer registers and calling the runsamples function is, however, not sufficient to activate mac-based sample play. Further analyzing the sample collect function, we realized that bits in the `psm_phy_hdr_param` register trigger and stop the sample collection. Hence, we systematically wrote various bit combinations into this register after calling the runsamples function until we observed the successful transmission of the samples stored in Template RAM.

To extract CSI, we found the `wlc_phydump_chanest` function by analyzing SoftMAC driver binaries. Chanest, respectively, channel estimation is commonly described as CSI. The function reads form physical layer tables to extract these values whenever the function is called. As the function is implemented in the driver or alternatively in the ARM firmware, it is not synchronized to frame receptions. Hence, the extracted values are often faulty as incoming frames overwrite old values on reception. We used the reference to the physical layer table to dump the CSI into the D11 core's shared memory on every frame reception. To this end, we had to modify the ucode. Transferring the CSI from shared memory to the ARM's RAM posed another challenge. As the D11 core cannot access the

Table 1: Broadcom's/Cypress' 802.11ac Wi-Fi chips currently supported by our firmware patches.

Chip	Firmware version	Used in	PHY/Core revision	Template RAM size	Signal length at 20 MHz BW	Currently supported features		
						Raw signal transmissions	CSI extraction	PHY-layer covert channel
4339a0	6.37.34.43	Nexus 5	6/46	256 KiB	1.64 ms	√	√	√
43455c0	7.45.154	Raspberry Pi B3+	20/54	256 KiB	1.64 ms	√		
4358a3	7.112.300.14	Nexus 6P	17/48	512 KiB	3.28 ms	√		

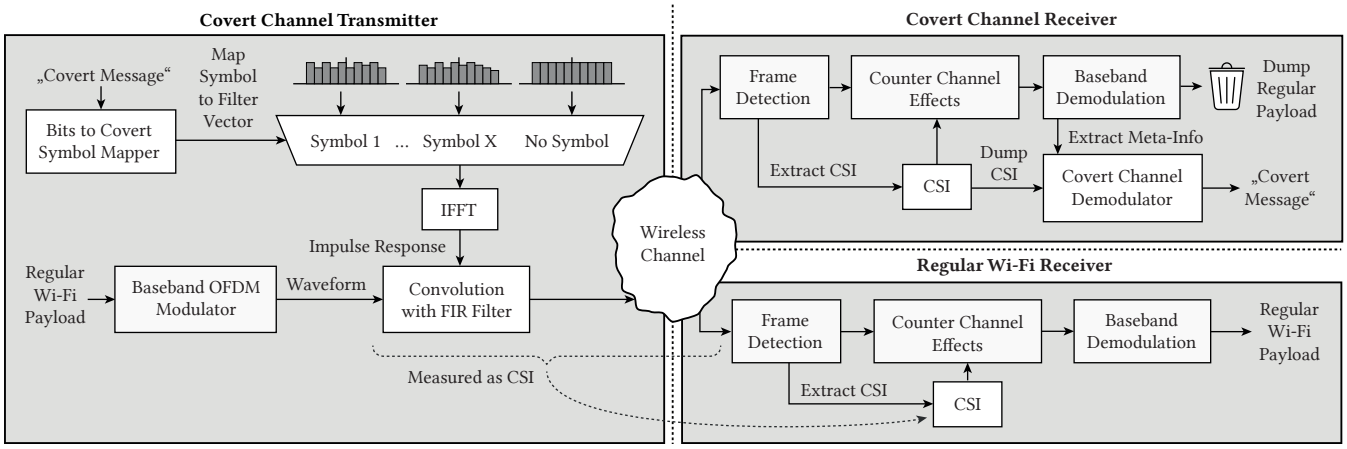

Figure 2: In our covert channel, the transmitter first maps message bits to symbols that select filters that we apply to outgoing frames by convolution in the time domain. A regular receiver estimates the transmit filter as part of the wireless channel so that its effect will be canceled. While a covert channel receiver may extract the hidden message from the measured channel state information.

RAM directly and triggering a copy from the ARM firmware is too slow, we would not be able to extract the CSI on a per-frame basis. Nevertheless, we observed that the DMA controller first copies a small header from shared memory and then the actual frame whenever a received frame is send to the host. Though, the header size is too small to hold the complete CSI dump, we realized that we can instruct the DMA controller to send a frame even though the receive FIFO is empty. In this case, only the header from shared memory gets copies and, in our case, contains the CSI dump.

4 COVERT CHANNEL DESIGN

Our covert channel relies on the ability of every Wi-Fi receiver to cope with the fading effects of the wireless channel to reconstruct the originally transmitted data symbols plus noise introduced during the transmission. We describe this receive operation in Section 3.1. By introducing an additional fading-like effect using a transmit filter, we can secretly embed additional information into each Wi-Fi frame without destroying the ability of a regular Wi-Fi receiver to correctly receive the frame. We illustrate our covert channel design in Figure 2. The covert channel transmitter may embed covert information for the covert channel receiver into Wi-Fi frames used during the communication with another node, the regular Wi-Fi receiver.

More formally, the effect of the wireless channel H_{sc} on a subcarrier sc can be linearly applied to the transmitted QAM-symbol $X_{sc}[k]$ of the k-th OFDM-symbol, resulting in a received symbol $Y_{sc}[k] = H_{sc} \cdot X_{sc}[k]$. H_{sc} is assumed to be constant during the transmission of one Wi-Fi frame. Hence, receivers extract it once per frame from the long-training field (LTF) resulting in the CSI measurement. Instead of transmitting the $X_{sc}[k]$ symbols directly, we can apply a filter F_{sc} to all transmitted symbols first, including those of the long-training field resulting in received symbols $Y_{sc}[k] = H_{sc} \cdot F_{sc} \cdot X_{sc}[k]$. As each Wi-Fi receiver estimates the CSI based on the channel effects applied to the LTF, it simply considers the transmit filter as part of the wireless channel $H'_{sc} = H_{sc} \cdot F_{sc}$ and automatically cancels its effect when decoding data signals. To simplify the equations, we combine variables depending on the subcarrier sc into column vectors, for example, $\vec{H} = \left(H_0 \ \cdots \ H_{\max(sc)}\right)^T$, leading to $\vec{Y}[k] = \text{diag}(\vec{H}) \cdot \text{diag}(\vec{F}) \cdot \vec{X}$.

To build a covert channel that does not disturb the regular Wi-Fi communication, we use differentiable filter vectors $\vec{F}_x$ as secret symbols. Hence, we can embed one secret symbol per frame. In the filter vectors, we can change amplitudes and phases of all transmitted symbols on all subcarriers. While changes in the amplitude are easily observable using spectrum analyzers, the detection of phase changes requires more thorough signal processing steps which impedes adversaries from easily detecting a covert transmission. Hence, we focus on modifying only the phases of particular subcarriers in transmitted frames. To mimic the behaviour of passing each Wi-Fi frame through an additional wireless channel when we apply the filter to embed covert symbols, we first convert our filter vector into the time-domain to create the filter's impulse response and then apply it by convolution with the Wi-Fi frames waveform. This increases the risk for inter-symbol interference (ISI), but it makes sure that this additional signal processing effect does not end on OFDM-symbol boundaries as it would be the case for filtering in the frequency domain. This avoids giving an adversary another feature he may look for when searching for irregularities that may imply the existence of a covert channel.

5 IMPLEMENTATION

Our work consists of three independent subsystems: (1) the raw signal transmitter (Section 5.1), (2) the channel state information (CSI) extractor (Section 5.2), and (3) the filter-based covert channel (Section 5.3). To generate the firmware patches for these subsystems, we use the Nexmon patching framework we created in our previous work [21, 22]. It allows to write our patches in C and apply them to the existing Wi-Fi firmware files. To ease reproducibility

of our results and to make our solutions available to the research community, we published our source code as described in Section 1.

5.1 Raw Signal Transmissions

As described in Section 3.2 and Section 3.3, we deduced the existence of the mac-based sample play capability by analyzing the runsamples function in combination with the sample collect function. After experimentally verifying that raw transmissions work in general, we developed the following methodology to ease the use of raw transmissions. All steps shown below can either be called directly from a firmware patch or by sending IOCTL messages from user space or the kernel of the host's operating system to the firmware. We give more details on communicating with the firmware in [21]. To prepare a transmission, we, first, copy raw IQ samples into an unused region of the Template RAM. Each sample consists of two signed 16-bit values for the inphase (I) and quadrature (Q) components. We can either generate the raw samples in the firmware or create them in MATLAB and use an IOCTL to copy them into Template RAM during runtime. Especially for long signals, it is not suitable to patch their samples into the firmware binary as they take up too much space in the patch memory region. In the second step, we set transmission gains to manual control and store the boundary addresses of our signal in the D11 core's sample play registers. Then, we are ready to trigger the transmission by activating `macbasedDACPlay`, triggering an RX-to-TX RF sequence and triggering the sample playback in the `psm_phy_hdr_param` respectively `SampleCollectPlayCtrl` register. This step can either be performed in the ARM core, or as a reaction to a time critical event in the D11 core (e.g., as a quick answer to receiving a frame). After finishing a transmission, the Wi-Fi chip continues to transmit a carrier wave and the D11 core is not able to receive any new frames. To solve this problem, we have to stop the transmitter and reset the clear channel assessment (CCA) in the baseband. Implementing this in the D11 core allows us to integrate our transmitter into regular Wi-Fi communications and react to incoming frames.

By publishing the source code for this project, we make sure that the research community can convert 802.11ac Wi-Fi chips into software-defined radios.

5.2 Channel State Information Extraction

To detect and decode our covert channel at the receiver side, we need to access the CSI on a per-frame basis, at least for those frames containing covert information. As described in Section 4, the CSI contains the amplitude and phase changes on each subcarrier introduced by the wireless channel. As this information is required to equalize the channel's effect on the transmitted symbols, the CSI is available in all OFDM-based Wi-Fi receivers. On Broadcom/Cypress cards, the physical layer extracts this information and stores it in a physical layer table, as described in Section 3.1 and Section 3.3. As the content of this table changes for every received frame, we have to read it during frame reception. The ARM microcontroller running the FullMAC firmware is not suitable for this task, as it processes only complete frames. Hence, we need to modify the code running on the D11 core's programmable state machine.

First, we analyze the maximum size of the CSI information data structure. The CSI itself consists of complex numbers $H_{sts,rx,sc}$ indicating phase and amplitude changes for each of the transmitted spacial streams sts, each receive antenna rx and each subcarrier sc. As each of the complex numbers is stored in 32-bit values, the total number of CSI bits per frame equals $32 \cdot sts \cdot rx \cdot sc$, respectively, $4 \cdot sts \cdot rx \cdot sc$ bytes. Each 802.11ac node can have between one and eight antennas, defining the range for sts and rx. The number of subcarriers depends on the used channel bandwidth and equals $\{64, 128, 256, 512\}$ for $\{20, 40, 80, 160\}$ MHz bandwidth. Only considering non-zero subcarriers, these numbers reduce to $\{56, 114, 242, 484\}$ used subcarriers. Hence, the smallest CSI array consists of $4 \cdot 1 \cdot 1 \cdot 56 = 224$ bytes, while the largest consists of $4 \cdot 8 \cdot 8 \cdot 484 = 121$ KiB.

As experimental platform for transmitting and receiving our covert channel, we use Nexus 5 smartphones. They feature a single Wi-Fi antenna and support 80 MHz bandwidth. Hence, the largest CSI consists of $4 \cdot 1 \cdot 1 \cdot 242 = 968$ bytes, which is roughly 69 times larger than a Wi-Fi acknowledgement frame (14 bytes). A key challenge for our system is to extract this amount of control information for every received frame. For the extraction, the D11 core needs to first copy the CSI from the physical layer CSI table into the D11 core's shared memory, which is limited in size and also stores variables that are required for regular Wi-Fi operation that should not be overwritten. After copying the information, we face the problem of transferring it to the RAM of the ARM core. Even though the ARM core can directly read from the shared memory, it is not guaranteed that the ARM core handles a frame reception sufficiently quick before the CSI in the shared memory is overwritten by the next incoming frame.

The optimal way for transferring CSI from shared memory to the ARM core's RAM would be to trigger a DMA transfer on the whole CSI containing memory region. To the best of our knowledge, the DMA controllers do not support this operation. Nevertheless, the D11 core prepends an additional header containing meta information to each received frame. This header is normally 28 bytes long and read from the shared memory during DMA transfers that copy the frame's payload into the ARM core's RAM. We realized that multiple of these DMA transfers can be triggered in a row. In this case, the transferred frames contain an empty payload, but the shared memory contents are copied in each transfer. By adjusting the meta header's start pointer after each transfer, we can copy the complete CSI from shared memory to the ARM core's RAM. Additionally, we managed to increase the size of those transfers from 28 to 64 bytes. Using four bytes for a header, 17 transfers are required to copy the whole 80 MHz CSI information. In the ARM core, we detect CSI transfers according to this header and can either process the received CSI directly or send it to the host using UDP frames.

By publishing the source code for this project, we make sure that the research community can extract CSI on smartphones.

5.3 Filter-based Covert Channel

To implement our covert channel, we have to prefilter outgoing frames to encode secret symbols that we can extract by evaluating the channel state information (CSI) at the covert channel receiver. We can implement the filter either in the time domain or in the frequency domain. In the time domain, we convolve the time-domain

signal $x(n)$ of every outgoing frame with the impulse response of the filter $f(n)$, so that the actually transmitted signal equals $x(n) * f(n)$. In the frequency domain, the convolution becomes a simple multiplication, so that we can filter every OFDM symbol $\vec{X}[k]$ by performing an element-wise multiplication with the filter vector $\vec{F}$, as described in Section 4. As OFDM-based Wi-Fi systems generate frequency-domain symbols first and then apply an inverse fast Fourier transform to create the time-domain signal, we can easily apply both filtering approaches. For filtering in the frequency domain, we have to change the OFDM modulator implementation and then apply the filter per OFDM symbol without affecting the cyclic prefix of the following OFDM symbol with the filter response. This approach would allow an attacker to detect the covert channel as it does not behave like a real wireless channel that creates inter-symbol interference (ISI) in the cyclic prefixes. For filtering in the time-domain, we can consider the OFDM modulator as a blackbox and better simulate the effect of a real wireless channel. Hence, we decided for an implementation in the time-domain.

Currently, we are not aware of a finite impulse response (FIR) filter in the transmit chain, that we could change on a per-frame basis to filter all outgoing frames that are efficiently created in the Wi-Fi baseband modulators. A possible solution is to implement the covert channel transmitter by handcrafting the complete Wi-Fi raw signals, filtering them and transmitting them with our raw signal transmission presented above. As this approach is very time consuming and does not support high frame rates, we decided for an optimized implementation that also works in real-time systems.

To this end, we assume that our covert channel transmitter is communicating with another node, for example, an access point. Instead of embedding the covert channel in the outgoing data frames—that are constantly changing—we embed the covert channel into the outgoing acknowledgements. As they are always addressed to the same communication partner (e.g., the access point), every frame contains exactly the same payload. Hence, we can generate prefiltered acknowledgement frame signals and store them in Template RAM. The higher the number of different prefiltered acknowledgements in Template RAM, the more covert symbols we can transmit. To additionally reduce the time for extracting CSI information, we can optimize our CSI extractor to only copy the CSI values of subcarriers used by our covert channel implementation.

For Wi-Fi frame generation, we use MATLAB's WLAN System Toolbox. It allows generating frames with arbitrary payload at all known modulation coding schemes (MCSs) and bandwidths. We use this toolbox to generate our acknowledgement frames and then apply various filters in the time domain to embed covert channel symbols. Then, we load the generated symbols into the Wi-Fi chip's Template RAM and transmit them either in a loop or triggered by the D11 core as a response to a frame received by the communication partner. Which implementation we use, depends on the experiment we want to run. While a transmission of raw signals in a loop can simply be triggered from the ARM core, the implementation in the D11 core is more complex and described below.

To answer with raw-signal acknowledgements from the D11 core, we have to intercept each frame reception after receiving the PLCP header and spin wait until we complete the reception of the MAC addresses. If we received a frame for us sent by our communication partner, we skip scheduling the transmission of a regular acknowledgement and set a variable that indicates to transmit a raw-signal acknowledgement. Then we wait until the frame is completely received, confirm that the frame check sequence (FCS) is correct and trigger a raw-signal transmission from Template RAM. To this end, we lookup start and end pointers in shared memory that indicate where the raw samples are stored in Template RAM and write them into the corresponding registers. Then we trigger the playback of these samples and perform the RX-to-TX sequencing to activate the transmitter. Then we spin wait until the end of the transmission and reset the clear channel assessment to stop the transmission and return the D11 core and the physical layer to a state where they can receive the next frame.

At the covert channel receiver, we dump the CSI information of all acknowledgement frames and additionally save the sequence number of the frame that was acknowledged to make sure that all covert symbols are received in the correct sequence. Then, we analyze the CSI dumps to extract the covert symbols. In a realistic setup the receiver has to synchronize on the transmitted symbols to identify where a covert channel transmission starts and where it ends. We implement a simple communication protocol for this purpose. We assume that our secret information is stored as bytes. Each byte is mapped to one or multiple covert symbols. To indicate the start and end of a covert byte, we use special covert symbols reserved as start and stop symbols. On the transmitter side, we store the secret messages as sequences of start and stop pointers in the shared memory that point to prefiltered acknowledgements containing different covert symbols. For each acknowledged frame that contains a new sequence number, the transmitter iterates over the shared memory entries to transmit the stored messages.

Last but not least, we define how covert symbols are modulated using filters to facilitate symbol extraction from the CSI at the covert channel receiver. We have three options to modulate symbols on each subcarrier: (1) changing the phase, (2) changing the amplitude, or (3) changing both of them. By influencing the amplitude, we change the average power on a limited number of subcarriers. Even though a regular multi-path channel may have a similar effect, the spectral shape of the transmitted frames may deviate from standard Wi-Fi frames, so that the existence of the covert channel is easily detected by looking at the power spectral density or the amplitude of the CSI on a line-of-sight channel. By modulating the phase, instead, we can achieve a better covertness of our channel, as its detection requires a look at phase changes in the CSI. Simply investigating transmitted signals with a spectrum analyser is not sufficient anymore to detect the existence of the channel.

6 EXPERIMENTAL EVALUATION

In our experimental evaluation, we study the performance of all three subsystems of our solution and directly discuss the experimental results. As environment we choose an apartment in a rural area with only low amount of Wi-Fi traffic so that our experiments are not affected by high levels of interference. Even by communicating between rooms and shielding the line-of-sight paths with a refrigerator, we only observed low multi-path effects in our CSI measurements. For experiments that do not focus on multi-path propagation, we simply stayed in one room. For influencing the

propagation characteristics of the wireless channel in repeatable fashion, we placed one device into a microwave oven and changed the opening angle of the microwave door. We document our different communication setups in Figure 3.

6.1 Raw-Signal-Transmission Experiments

To evaluate the raw signal transmission capabilities, we measure how well Wi-Fi frames transmitted from raw samples stored in the Template RAM can be received by an off-the-shelf Wi-Fi node, in comparison to receiving the same frames transmitted through the regular Wi-Fi modulation chain. To this end, we first generate acknowledgement frames at all 802.11a/g rates in MATLAB and store the signals in a format that we can directly load into the Template RAM by executing a script in the smartphone's user space after loading our patched Wi-Fi firmware. Due to the size of the raw signals, we cannot fit them directly into the firmware patch. In the firmware, we trigger the transmission of the raw signals every 3 ms and after a short break of 1 ms, we directly inject an acknowledgement frame through the regular frame transmission path. This frame is encoded with the same modulation settings and contains a similar content as our raw Wi-Fi frames.

Our experimental results show that frames transmitted through raw signal transmission have a similar reception performance as real frames. This is the expected result as it should not make a difference whether IQ samples from the Template RAM or the dedicated OFDM modulator are injected into the analog front-end by using digital-to-analog converters. Nevertheless, raw transmissions can fail on crowded channels. We observed that sometimes only a carrier wave is being transmitted instead of the raw signal. This effect happened very randomly in our experiments but seems to be avoidable on unoccupied channels. We illustrate this effect in Figure 4. On the left we show a transmission, where only the carrier is observable and on the right we show a correctly transmitted acknowledgement frame. Even though, this bug may lead to the discovery of the covert channel in a practical system, it is negligible in the proof-of-concept implementation we performed for this work. To eliminate an influence of this bug on our experiments, we concentrate on working only on otherwise empty Wi-Fi channels.

Additionally, in Figure 4 we observe that a small carrier signal is visible before and after the transmitted acknowledgement frames. This is due to the fact, that we first perform the RX-to-TX sequencing and then start the sample playback from the Template RAM. By first triggering sample playback of a signal that starts with a suitable number of zeros and then triggering the RX-to-TX sequencing, we can start playing samples as soon as the transmitter is ready. To get rid of the small carrier after the signal transmission, we need to initiate the CCA reset directly at the end of the raw transmission.

6.2 Limitations in throughput and latency

To analyze the throughput for transferring samples between the ARM's RAM and the Template RAM, we implemented a copy loop in the ARM firmware. On BCM4339 Wi-Fi chips, we achieved throughputs of 52.9 MBps for writing to and 25.0 MBps for reading from Template RAM. As 160 MBps are required to continuously provide the minimum of 40 MSps, our transmitter implementation currently only supports frame based transmissions and takes about

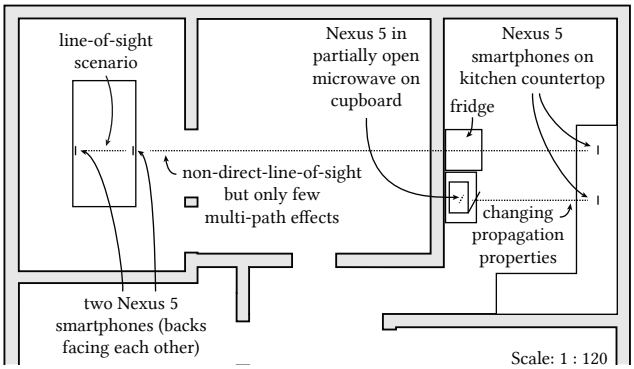

Figure 3: Experimental setup in an apartment in a rural environment with no other Wi-Fi traffic on channel 122 and 120. Placing the transmitter in the partially open microwave increases multi-path effects. All smartphones are installed on car mount holders to enhance the antenna radiation characteristics.

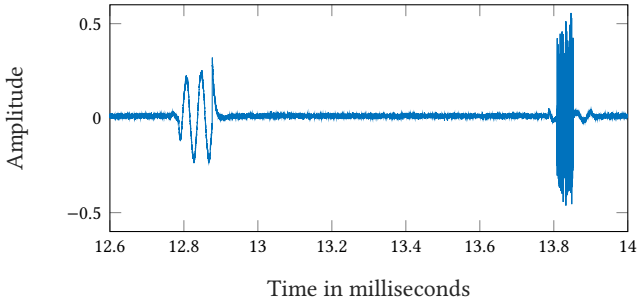

Figure 4: Generally raw transmissions play back the IQ samples from the Template RAM resulting in the signal on the right. Especially, on crowded channels it can randomly happen that only a carrier is transmitted but no other signal (left).

three times the signal length for transferring samples into Template RAM. For our covert channel implementation, we need to answer a frame reception with a handcrafted acknowledgement. To evaluate the latency of performing this task in the ARM processor, we activated monitor mode and triggered the IQ sample transfer into Template RAM as well as the transmission of those samples on every incoming frame. With another Wi-Fi node, we measured the reception time stamps of Wi-Fi frames and their corresponding handcrafted acknowledgements. Our results show, that the time between the end of a frame's reception and the beginning of the acknowledgement is roughly 614 μs. This latency is much higher than the timing requirements of 802.11 systems and, hence, renders real-time experiments almost impossible. Only in special cases, where pregenerated frames can be stored in the Template RAM and their transmission is triggered by the D11 core, strict timing requirements can be met—as demonstrated by our covert channel implementation.

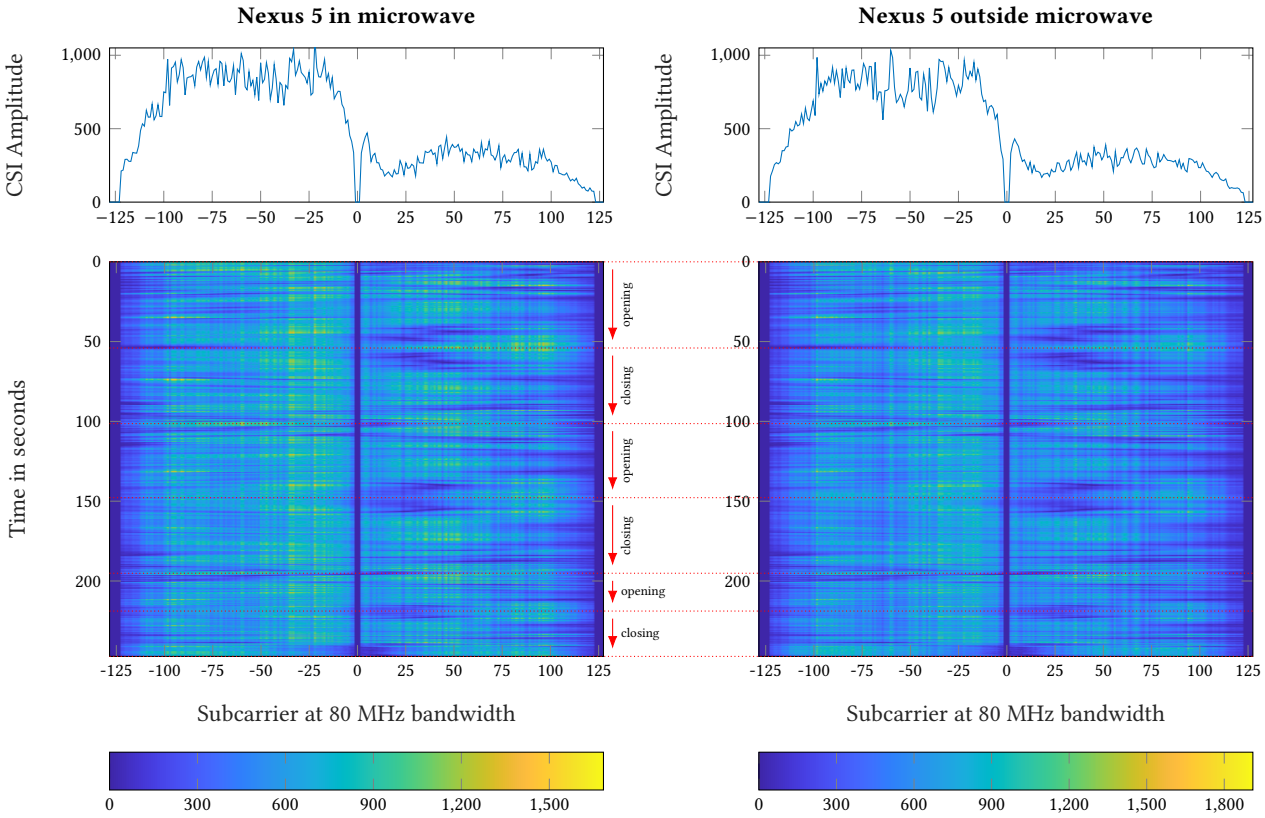

Figure 5: Continuous bi-directional channel state information measurement between two Nexus 5 smartphones, one in a microwave oven, the other one outside, while opening and closing the microwave oven door three times. The measurement is taken at channel 122 with 80 MHz bandwidth in the 5 GHz band. At the top, we illustrate the amplitudes of the tenth CSI measurement, followed by a waterfall diagram with 10 measurements per second. The channel reciprocity is clearly observable.

6.3 CSI-Extraction Experiments

As presented in Section 2, extracting channel state information on off-the-shelf devices can lead to various interesting applications. As evaluating our CSI extractor against all existing solutions is out of scope of this work, we consider a simple example applications that demonstrates the performance of our extractor. To this end, we setup two Nexus 5 smartphones in the kitchen of an apartment. One goes into our microwave oven, the other one on the kitchen countertop facing the microwave oven as illustrated in Figure 3. Even with a closed microwave-oven door, the two nodes can exchange Wi-Fi frames on the 80 MHz wide channel 122 in the 5 GHz band. Nevertheless, the position of the oven door heavily influences the propagation characteristics of the wireless channel between our two smartphones. We intend to illustrate this effect, by extracting the channel state information on both of the two nodes.

We configure the node outside the microwave oven to transmit frames every 100 ms directly from the firmware. The node in the microwave oven receives those frames, extracts the CSI in the D11 core and sends it up to the ARM core. After the CSI is fully received, the ARM core stores it in a UDP datagram and forwards it to the user space, where we dump it using tcpdump. Additionally, the ARM core crafts a new frame and sends it back to the other smartphone, which also extracts the CSI and forwards it to user space for dumping. While the two nodes exchange frames and dump CSI values, we open and close the microwave oven door three times. The first two times rather slowly, the third time a bit faster.

After running the experiments, we collect the frame dumps from the two smartphones. Then, we analyse the per frame CSI information in MATLAB and plot the magnitudes of the CSI for every subcarrier. We illustrate the result in Figure 5. At the top, we plot the CSI of the tenth exchanged frame for both of the two nodes. Below, we plot the magnitudes of the following CSI frames as a waterfall diagram. Comparing the two waterfall diagrams as well as the single plots clearly shows the similarity of the CSI measurements in both directions. This lets us imply the reciprocity of the wireless channel, which is an expected result. Additionally, we can observe repeating pattern in the waterfall diagrams. They repeat whenever we repeat the action of opening and closing the oven door. Regarding both actions, the pattern during closing the door is mirrored into the pattern of opening the door. Which indicates a very stable wireless environment, in which the wireless channel between the two smartphones is mainly influenced by the position of the oven door.

6.4 Covert-Channel Experiments

To evaluate the covert channel performance, we look at three different aspects. First, we evaluate how embedded covert symbols influence the frame reception at normal Wi-Fi receivers. Second, we measure covert symbol detection ratios at the covert channel receiver. Third, we perform a realistic message exchange using our covert channel and demonstrate that it is practical.

For our evaluation, we generate 16 different covert symbols (0000_2, 0001_2 ... 1111_2). Each of them is a filter vector in the frequency domain that changes the phases of the four subcarriers $8+s$, $11+s$, $54-s$, and $58-s$ by 170°, 190°, 170°, and 190°, where $s \in [0, \ldots, 15]$ is the symbol index. We transform these vectors into the time domain by using an IFFT to generate the filters' impulse responses that we can apply to Wi-Fi frame signals by convolution. Our frames are 802.11g modulated at all eight modulation coding schemes (MCSs), resulting in the bit rates from 6 Mbps to 54 Mbps. We generate those frames in MATLAB and filter each of them with our 16 covert symbols, which results in 128 acknowledgement frames with embedded covert symbols that we use for the experiments described below. To ease our analysis, we also write a covert symbol identifier into the last two bytes of the frames' MAC addresses. Obviously, for real covert channel operation, this must be avoided.

In our first experimental setup, we place two Nexus 5 smartphones in a distance of 1 m on a table top with the backs facing each other (see Figure 3). We use the phone on the right as covert channel transmitter. For each sub-experiment, we first load the raw samples of one of the generated acknowledgement frames into the Template RAM and then start a timer that triggers the transmission of 10 000 raw frames—one per millisecond. We use the second Nexus 5 smartphone on the left as both off-the-shelf Wi-Fi receiver and covert channel extractor to simplify the experimental setup without loss of generality. In a realistic scenario Wi-Fi receiver and covert channel extractor would be two different nodes to hide the intended covert channel receiver. To this end, the phone always dumps the received acknowledgement frame as well as the channel state information into a pcap file. After repeating the experiment for all 128 pre-generated frames, we analyze the pcap contents.

First, we analyze the effect of the embedded covert symbols to the regular Wi-Fi receiver. We count frames per MAC address. As this address contains an identifier of the embedded covert symbol, we directly know which frames are modified and how they were pre-filtered. In Figure 6, we present the results. Each row illustrates the number of received frames during the eight experiments—one for each MCS. The first row is the result without covert symbols. Here, the maximum of 10 000 frames per sub-experiment was almost reached. The next six rows belong to covert symbols that have no measurable influence on the reception performance for MCSs of up to 24 Mbps. They are, hence, a good choice for embedding covert symbols without raising suspicions at covert channel detectors that take frame error rates into account. For the other rows below, the influence on regular Wi-Fi receivers increases which renders those symbols inadequate for staying undiscovered.

Secondly, we analyze how well the intended receiver of the covert channel can detect and differentiate the different symbols. To this end, we analyze the CSI dumps and extract the phase at

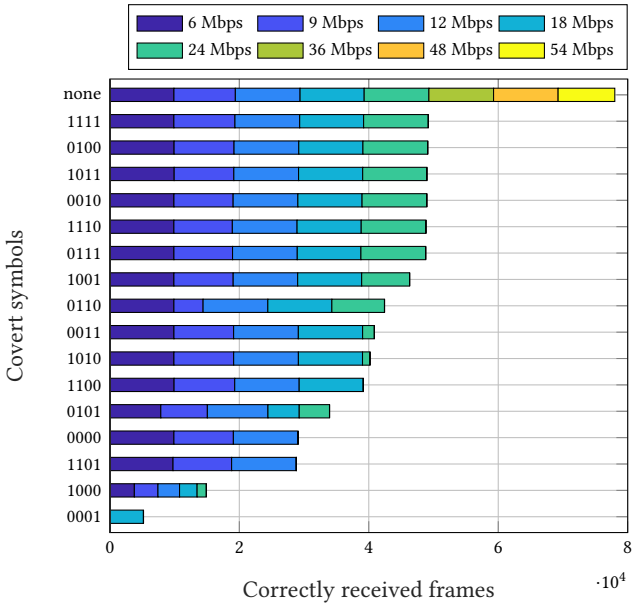

Figure 6: Frame reception rates at off-the-shelf Wi-Fi receiver with different covert channel symbols that each flip the phases on four subcarriers.

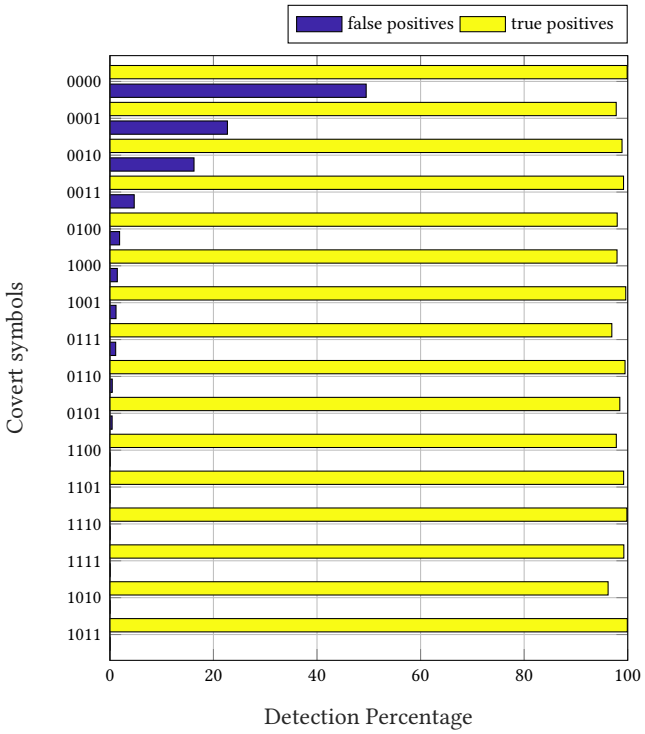

Figure 7: Detection rates at the covert channel receiver. False positives occur when a covert symbol is detected in an unmodified frame. True positives occur when the embedded symbol was detected.

all subcarriers. As all of our covert symbols are encoded by changing the phases at four subcarriers by roughly 180°, we search for jumps at the expected positions at the receiver's CSI phases and calculate a likeliness value for each symbol to be contained in the received frame. At the end, we choose the symbol with the highest likeliness value as the one we extract for the corresponding frame. If the detected symbol equals the embedded symbol we have a true positive. In Figure 7, we illustrate the percentage of achieving true positives for each symbol. Here, we realize that all different symbols achieve true positive rates reaching 100 percent. Which makes them all usable in a practical covert channel. Additionally, we also analyzed the percentage of false positives. Those are symbols that we detect even though no symbols were embedded in the frame. Some symbols are more likely to be detected in this case. In general, even these symbols are usable in practical covert channels, but they should be used to encode data rather than the beginning of a message. Otherwise, we risk to detect many wrong message starts.

Combining the results of the two measurements, we choose the six symbols 1111_2, 1011_2, 0100_2, 0010_2, 1110_2 and 0111_2 as covert channel symbols. As 1111_2 does not result in false positives, we choose it as message start symbol. 1011_2 is our message stop symbol and the other four symbols are data symbols, each encoding two bits: 00_2, 01_2, 10_2 and 11_2. For our third experiment, we use the selected symbols to transfer bytes over the covert channel. To signal the receiver that a data transmission starts, we first send the message start symbol and then encode our bits with data symbols followed by a stop message symbol. We chose to place one byte per message, which results in six symbols that need to be transmitted. Of course, the number of bits can be varied to more efficiently use the covert channel as long as both receiver and transmitter use and expect the same fixed number of data symbols per message. To transmit the messages, we embed them in acknowledgement frames that we can generate for our communication partner in advance. Then we use our ucode modifications to trigger the transmission of the correct acknowledgement corresponding to the covert symbol that should be transmitted. The start pointers for the acknowledgements are stored in the shared memory and written by the ARM core after mapping the desired messages to covert symbols.

In general, our covert channel could communicate normally with a regular Wi-Fi node and embed the covert symbols into the acknowledgements for this node. The actual covert channel receiver could, however, be another node that eavesdrops on the communication between the two nodes to capture the acknowledgements and dump the CSIs to extract the covert symbols. Using this setup, the covertly communicating nodes could conceal the communication partners. For our experiments, we simplify this setup by using the same node as communication partner for the regular Wi-Fi transmissions and as covert channel receiver. In this setup, the Wi-Fi communication partner simply injects frames to the covert channel transmitter. The latter checks for an expected MAC address and transmits a raw acknowledgement with an embedded covert symbol. The covert channel receiver captures this frame, dumps the CSI and extracts the covert symbol and then tries to reconstruct the covert messages, each consisting of one byte.

We ran experiments on our covert channel implementation in three indoor scenarios that we illustrate in Figure 3. First, we keep two Nexus 5 smartphones close together in the line-of-sight setup

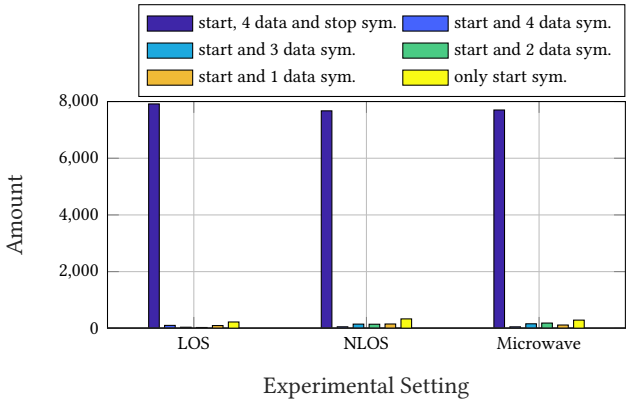

Figure 8: Results of transferring one byte in four covert symbols plus start and stop symbols in a line-of-sight (LOS), non-line-of-sight (NLOS) and microwave environment with changing wireless channel characteristics.

on the living room table. Then we evaluate a non-line-of-sight setup by keeping one smartphone on the living room table and placing the other on the kitchen countertop so that a direct line-of-sight connection is impeded by a fridge. Both setups simply evaluate the influences of multi-path effects on our covert transmission. In the third setup, we check whether our implementation can cope with quickly changing wireless propagation characteristics, by placing one smartphone in a microwave oven and another one on the kitchen countertop similar to our CSI extraction experiment in Section 6.3. By quickly opening and closing the microwave oven door, we constantly change the wireless channel and thereby the measured CSI.

In Figure 8, we illustrate the results of the three experiments. For each experiment, we count how many correct symbols in a row we were able to extract leading to either completely correct message receptions, partially correct message receptions or wrong message receptions. The total possible number of transmitted messages would be 10 000, as we transmitted 60 000 Wi-Fi frames expecting a covert symbol containing acknowledgement. The missing acknowledgements at the receiver were either not correctly received or not transmitted due to a reception problem of the ingoing Wi-Fi frame at the covert channel transmitter. The first bar in Figure 8 shows how many messages we extracted completely correct. In all three experiments, most of the extracted covert symbols resulted in complete messages. Only few messages were missing symbols to reconstruct the full message. Overall, these results show that our covert channel is indeed practical even in non-line-of-sight environments as well as those with constantly changing wireless propagation characteristics due to movement in the environment (in this case the microwave oven door). Further error correction of the received messages is left for upper layers that might even enhance confidentiality of the transmitted covert symbols by applying cryptography with a shared key between the covert channel communication partners. This would at least hinder leaking the covertly transmitted information in case an adversary discovers and decodes this covert channel.

Regarding performance, with the selected symbols, we can transmit 2 bits per Wi-Fi frame. Four frames make up one byte that is surrounded by one message start and one message stop symbol. Hence, six acknowledgement frames are used to transmit one byte. An acknowledgement frame modulated with 24 Mbps is 28 μs long. Assuming that such frames are transmitted continuously, roughly 35 714 frames could be transmitted per second, which results in a gross covert channel transmission rate of 5 952 Bps. Of course, net rates are smaller as interframe spacing and data frames that result in the transmission of acknowledgements need to be taken into consideration.

7 DISCUSSION

In this work, we demonstrated that off-the-shelf Wi-Fi chips have much more capabilities than officially advertised by vendors. Using Wi-Fi chips in a way only software-defined radios could be used before, we open new scenarios for researchers that can transform ubiquitously available mobile devices such as smartphones into general purpose radio transmitters in the 2.4 and 5 GHz bands. With the ability to store raw IQ samples in a relatively large buffer it is now possible to add physical layers from other standards or even test new ones: this enhances cross-technology communication to an unprecedented degree of flexibility. The great availability of smartphones also enables the creation of large software-defined radio testbeds built on devices that are cheap, mobile, and always around the end users.

Regarding the analysis of the environment, the ability to dump uncompressed channel state information on a per-frame basis with smartphones allows to either enhance already existing solutions by adding mobility or even implement new applications. Compared to already available CSI extraction firmwares for other off-the-shelf Wi-Fi chips, we support 802.11ac frames with currently 80 MHz bandwidth, which gives at least twice as much insight into wireless propagation characteristics as existing solutions. Additionally, the source code of all firmware patches developed for this work are publicly available to allow fellow researchers to customize our applications for their particular research needs. For example, by preprocessing CSI directly in the Wi-Fi chips ARM processor and delivering only aggregated information to an app running on the smartphone. The similarity of all Broadcom Wi-Fi chips also allows to port our solutions to other devices that have more antennas and offer even wider bandwidth of up to 160 MHz.

Last but not least, our example application—that combines both the abilities to transmit raw signals and extract channel state information—is the first physical layer covert channel where both the transmitter and the receiver are implemented on an off-the-shelf Wi-Fi chip that is installed in smartphones. Covert channels such as ours can help to exchange messages between wirelessly enabled users without even revealing that a covert transmission is ongoing as well as who the receiver of the messages is. On the other side, adversaries can also use covert channels to secretly extract private information from a protected system. Even if intrusion detection systems may analyze the regular Wi-Fi frame contents, they most likely miss irregularities on the physical layer. To counter these information leakage attacks, security experts need to know how covert channels could be implemented to be able to at least detect the existence of an information leak so that it can be found and closed. Hence, for both applications, it is important to discuss new ways of implementing covert channels.

8 FUTURE WORK

Besides supporting more chip models to reach an even wider range of potential users, there are multiple possibilities to enhance and extend our work. By getting a better understanding of the direct memory access controllers, one may transfer data directly from host to Template RAM with a speed high enough to refill half of the buffer while the respectively other half is being transmitted. This would allow continuous signal transmissions requiring a throughput of 160 MBps for 20 MHz bandwidth. This should be achievable using the Wi-Fi chip's PCI Express interface. One could reduce the required throughput by upsampling signals on the fly and finding a transmit filter to limit the bandwidth and thereby remove aliases.

Due to the flexibility of injecting raw IQ samples into the 2.4 and 5 GHz transceivers, one can use Wi-Fi chips in commodity devices to directly send messages to other technologies operating in these frequency bands (cross-technology communication). Examples are ZigBee devices, amateur radio systems, satellite uplinks, drones and toy cars, human interface devices such as keyboards, baby monitors, video surveillance cameras and much more. By additionally activating the Wi-Fi chips' sample capture capabilities one could even implement bi-directional communication with other technologies. Due to the low cost of the Raspberry Pi, our system can also be used for teaching practical wireless systems in classes at universities and schools.

Using CSI extraction on mobile devices will likely lead to new mobile applications such as indoor localization or the surveillance of the environment by turning the Wi-Fi chip into a sensor that can detect movements in a device's vicinity.

9 CONCLUSION

In this work, we managed to send raw radio signals in the 2.4 and 5 GHz bands in a software-defined-radio-like fashion using Wi-Fi chips of off-the-shelf smartphones. To this end, we reverse engineered and extended the Wi-Fi chip's firmware. With a second firmware patch, we extracted uncompressed per frame channel state information (CSI) from Wi-Fi transmissions at 80 MHz bandwidth on Nexus 5 smartphones. Using both the raw signal transmission capabilities and the CSI extractor, we implemented and evaluated a practical physical layer covert channel that directly runs on Wi-Fi chips. The covert channel uses a transmit filter to embed covert symbols into the phases of OFDM-based Wi-Fi frames in a way that a receiver can extract the covert symbols from CSI measurements. Our results show that our implementation works even in non-line-of-sight scenarios and environments with quickly changing wireless propagation characteristics.

10 ACKNOWLEDGMENTS

This work has been funded by the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 "MAKI – Multi-Mechanism-Adaptation for the Future Internet", by LOEWE NICER, LOEWE CASED, and by BMBF/HMWK CRISP. Many thanks go to our shepherd, Lin Zhong.

REFERENCES

[1] I. E. Bagci, U. Roedig, I. Martinovic, M. Schulz, and M. Hollick. 2015. Using Channel State Information for Tamper Detection in the Internet of Things. In *Proc. of the 31st Annual Computer Security Applications Conference (ACSAC '15)*. ACM, 131–140.

[2] Jiska Classen, Matthias Schulz, and Matthias Hollick. 2015. Practical covert channels for WiFi systems. In *2015 IEEE Conference on Communications and Network Security (CNS)*. 209–217.

[3] Mango Communications. 2017. WARP Project. http://warpproject.org

[4] Aveek Dutta, Dola Saha, Dirk Grunwald, and Douglas Sicker. 2013. *Secret Agent Radio: Covert Communication through Dirty Constellations*. Springer Berlin Heidelberg, Berlin, Heidelberg, 160–175.

[5] Ettus Research, A National Instruments Company. 2010. https://www.ettus.com.

[6] C. G. Girling. 1987. Covert Channels in LAN's. *IEEE Transactions on Software Engineering* SE-13, 2 (1987), 292–296.

[7] Iwona Grabska and Krzysztof Szczypiorski. 2013. Steganography in WiMAX networks. In *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. 20–27.

[8] Iwona Grabska and Krzysztof Szczypiorski. 2014. Steganography in Long Term Evolution Systems. In *2014 IEEE Security and Privacy Workshops*. 92–99.

[9] Szymon Grabski and Krzysztof Szczypiorski. 2013. Steganography in OFDM Symbols of Fast IEEE 802.11n Networks. In *2013 IEEE Security and Privacy Workshops*. 158–164.

[10] Great Scott Gadgets. 2009. HackRF One. http://greatscottgadgets.com/hackrf/.

[11] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. 2011. Tool Release: Gathering 802.11N Traces with Channel State Information. *SIGCOMM Comput. Commun. Rev.* 41, 1 (Jan. 2011), 53–53.

[12] Zaid Hijaz and Victor S. Frost. 2010. Exploiting OFDM systems for covert communication. In *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*. 2149–2155.

[13] Rizky Pratama Hudhajanto, I Gede Puja Astawa, and Amang Sudarsono. 2016. Covert Communication in MIMO-OFDM System Using Pseudo Random Location of Fake Subcarriers. *EMITTER International Journal of Engineering Technology* 4, 1 (2016).

[14] Z. Jiang, J. Zhao, X. Y. Li, J. Han, and W. Xi. 2013. Rejecting the Attack: Source Authentication for Wi-Fi Management Frames using CSI Information. In *Proc. of the 32nd IEEE International Conference on Computer Communications (INFOCOM '13)*. IEEE, 2544–2552.

[15] Swarun Kumar, Diego Cifuentes, Shyamnath Gollakota, and Dina Katabi. 2013. Bringing Cross-layer MIMO to Today's Wireless LANs. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. 387–398.

[16] Butler W. Lampson. 1973. A Note on the Confinement Problem. *Commun. ACM* 16, 10 (1973), 613–615.

[17] Zhijun Li and Tian He. 2017. WEBee: Physical-Layer Cross-Technology Communication via Emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. 2–14.

[18] Lime Microsystems. 2017. LimeSDR. http://www.limemicro.com/products/software-defined-radio/.

[19] F. Ricciato, S. Sciancalepore, F. Gringoli, N. Facchi, and G. Boggia. 2018. Position and Velocity Estimation of a Non-cooperative Source From Asynchronous Packet Arrival Time Measurements. *IEEE Transactions on Mobile Computing* (2018), 1–1. https://doi.org/10.1109/TMC.2018.2792443

[20] Matthias Schulz, Francesco Gringoli, Daniel Steinmetzer, Michael Koch, and Matthias Hollick. 2017. Massive Reactive Smartphone-Based Jamming using Arbitrary Waveforms and Adaptive Power Control. In *Proc. of the ACM Conference on Security and Privacy in Wireless & Mobile Networks (WiSec) 2017*. Boston, USA.

[21] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2017. Nexmon: Build Your Own Wi-Fi Testbeds With Low-Level MAC and PHY-Access Using Firmware Patches on Off-the-Shelf Mobile Devices. In *Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization (WiNTECH '17)*. 59–66.

[22] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2017. Nexmon: The C-based Firmware Patching Framework. https://nexmon.org

[23] K. Szczypiorski. 2003. HICCUPS: Hidden communication system for corrupted networks. In *In Proc. of: The Tenth International Multi-Conference on Advanced Computer Systems ACS'2003, October 22-24, 2003 Miedzyzdroje*. 31–40.

[24] Krzysztof Szczypiorski and Wojciech Mazurczyk. 2010. Hiding Data in OFDM Symbols of IEEE 802.11 Networks. In *2010 International Conference on Multimedia Information Networking and Security*. 835–840.

[25] Kun Tan, Jiansong Zhang, Ji Fang, He Liu, Yusheng Ye, Shen Wang, Yongguang Zhang, Haitao Wu, Wei Wang, and Geoffrey M. Voelker. 2009. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. 75–90.

[26] Manfred Wolf. 1989. *Covert channels in LAN protocols*. Springer Berlin Heidelberg, Berlin, Heidelberg, 89–101.

[27] Yaxiong Xie, Zhenjiang Li, and Mo Li. 2015. Precise Power Delay Profiling with Commodity WiFi. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. 53–64.