PhyAuth: Physical-Layer Message Authentication for ZigBee Networks

Ang Li and Jiawei Li, *Arizona State University;* Dianqi Han, *University of Texas at Arlington;* Yan Zhang, *The University of Akron;* Tao Li, *Indiana University–Purdue University Indianapolis;* Ting Zhu, *The Ohio State University;* Yanchao Zhang, *Arizona State University*

https://www.usenix.org/conference/usenixsecurity23/presentation/li-ang

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

PhyAuth: Physical-Layer Message Authentication for ZigBee Networks

Ang Li
Arizona State University

Jiawei Li
Arizona State University

Dianqi Han
University of Texas at Arlington

Yan Zhang
The University of Akron

Tao Li
Indiana University–Purdue University Indianapolis

Ting Zhu
The Ohio State University

Yanchao Zhang
Arizona State University

Abstract

ZigBee is a popular wireless communication standard for Internet of Things (IoT) networks. Since each ZigBee network uses hop-by-hop network-layer message authentication based on a common network key, it is highly vulnerable to packet-injection attacks, in which the adversary exploits the compromised network key to inject arbitrary fake packets from any spoofed address to disrupt network operations and consume the network/device resources. In this paper, we present PhyAuth, a PHY hop-by-hop message authentication framework to defend against packet-injection attacks in ZigBee networks. The key idea of PhyAuth is to let each ZigBee transmitter embed into its PHY signals a PHY one-time password (called POTP) derived from a device-specific secret key and an efficient cryptographic hash function. An authentic POTP serves as the transmitter's PHY transmission permission for the corresponding packet. PhyAuth provides three schemes to embed, detect, and verify POTPs based on different features of ZigBee PHY signals. In addition, PhyAuth involves lightweight PHY signal processing and no change to the ZigBee protocol stack. Comprehensive USRP experiments confirm that PhyAuth can efficiently detect fake packets with very low false-positive and false-negative rates while having a negligible negative impact on normal data transmissions.

1 Introduction

ZigBee is an IEEE 802.15.4-based specification very popular for Internet of Things (IoT) networks [13]. The ZigBee stack architecture comprises four layers from low to high: the Physical layer (PHY), medium access control layer (MAC), network layer (NWK), and application layer (APL) [3]. The APL layer consists of the application support sublayer (APS) and the ZigBee device objects (ZDO). PHY and MAC layers are defined in the IEEE 802.15.4 standard [7], and NWK and APL layers are defined in the ZigBee specification [3].

A ZigBee network consists of three device types: coordinator, router, and end device. Each ZigBee network has exactly one coordinator and can have many routers and end devices.

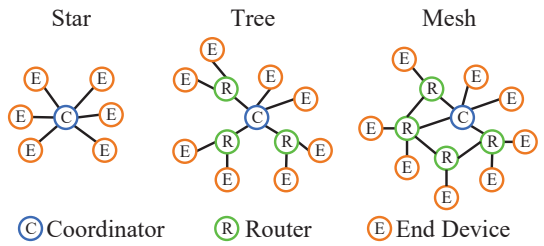


Figure 1: ZigBee network topologies.

The coordinator acts as a central node responsible for managing the ZigBee network. Routers can route traffic between different devices. End devices (e.g., wireless sensors) can only transmit/receive a message to/from their parent nodes (routers or the coordinator). As shown in Fig. 1, a ZigBee network can operate in three network topologies and support up to 65,000 nodes [3]. ZigBee supports multi-hop mesh networking based on the AODV (Ad-hoc On-demand Distance Vector) routing protocol [27]. ZigBee packets comprise unicast transmissions between two APL peer entities that can be multi-hop away, network-wide or one-hop broadcast transmissions, and multicast transmissions to a group of nodes. ZigBee has been widely used in many mission-critical contexts, such as hospitals and healthcare automation, critical-infrastructure monitoring and management, industrial/home/building automation, personnel and asset tracking, smart-city sensing, and military/defense applications [11]. For example, the UK government uses ZigBee mesh networks to connect smart meters in homes to the utility network [14]. In addition, smart factories use ZigBee-enabled sensors to monitor and manage the entire manufacturing process [17, 21]. Moreover, ZigBee networks are used to control the routes of multiple automatic guided vehicles in smart factories or warehouses [12].

ZigBee defines security mechanisms at the NWK and APS layers based on "link" keys and a "network" key [3]. In particular, unicast frames between APL peer entities are secured with a 128-bit link key shared by the source and destination, while broadcast/multicast frames and all network-layer frames

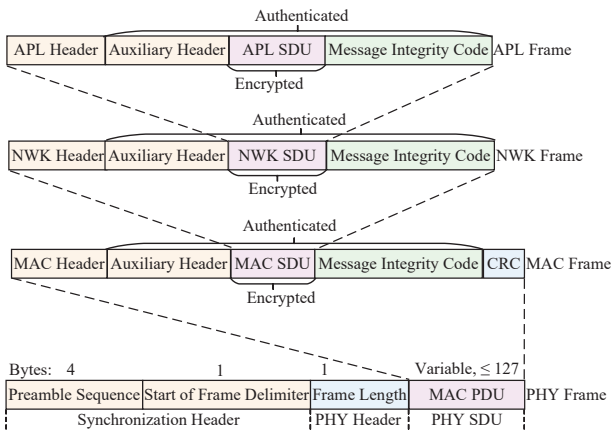


Figure 2: The IEEE 802.15.4/ZigBee frame structure with all possible security features.

(e.g., routing commands) are secured with a 128-bit network key shared amongst all devices in the network. As shown in Fig. 2, every NWK frame is encrypted and/or authenticated with a Message Integrity Code (MIC) based on the network key and the 128-bit AES-CCM algorithm.[1] An APS frame can also be encrypted and/or authenticated with a MIC based on 128-bit AES-CCM with the network key or a link key according to the frame type and application requirements. ZigBee achieves *hop-by-hop* message authentication by letting each node use the network key to verify the MIC of each incoming NWK frame before any further processing (e.g., forwarding to the next hop). In contrast, the APS MIC is only verified at the APS entity of the end-to-end destination.

Hop-by-hop message authentication based on the common network key makes ZigBee networks very vulnerable to *packet-injection attacks*. In such an attack, the adversary first acquires the authentic network key by compromising any node in the large network. In the current ZigBee security architecture, a ZigBee device is considered legitimate by its neighbors as long as it shows the knowledge of the correct network key by appending an authentic MIC in each forwarded/originated NWK frame. So the adversary can exploit the compromised network key to inject arbitrary fake NWK frames with forged content but a valid MIC from spoofed legitimate or even random device addresses, which can all evade hop-by-hop message authentication. NWK frames can be classified into command frames for routing and network management and data frames carrying APS-layer broadcast/unicast/multicast data messages. Since NWK command frames and NWK data frames containing APS-layer broadcast data are only authenticated with the network key, fake packets targeting these frame types can propagate throughout the entire network to be falsely accepted by every node, leading to severe disruption of network operations and quick depletion of device batteries. Even if some NWK data frames such as those carrying APS-layer unicast data can be additionally authenticated by an APS MIC based on a non-compromised end-to-end link key, fake packets involving these frame types can only be detected by the APL entity at the final destination and would have consumed massive network resources for being relayed along multi-hop paths. An intuitive countermeasure against such packet-injection attacks is to replace the common network key with unique node-dependent keys to (re)generate and verify the NWK MICs at every hop towards the destination. This plausible defense would involve a major change to the NWK layer of the ZigBee specification and is thus impractical.

In this paper, we present **PhyAuth**, a PHY hop-by-hop message authentication framework that complements the current ZigBee NWK hop-by-hop message authentication method. The key idea of PhyAuth is to let each ZigBee transmitter embed into its PHY signals a **PHY one-time password** (called POTP) derived from a device-specific secret key. An authentic POTP serves as the transmitter's PHY transmission permission for the corresponding NWK frame. A *verifier* authenticates a ZigBee transmitter by detecting and verifying the POTP from its PHY signals. Verifiers can be normal ZigBee receivers or dedicated intrusion detection systems (IDS) not engaged in normal ZigBee communications. If a valid POTP cannot be detected, verifiers drop the corresponding NWK frame without any further processing; they can also send an alert message to the network administrator which can physically locate and remove illegitimate transmitters. POTP generation and verification use any standard cryptographic hash function implemented in software or hardware.

PhyAuth includes three methods—**VarChip**, **VarAmp**, and **VarPhase**—that explore different features of ZigBee PHY frames (Fig. 2) to embed a POTP. In particular, ZigBee uses the IEEE 802.15.4 PHY layer which explores Direct Sequence Spread Spectrum (DSSS) to improve interference and noise resilience (§2.2). Each 4-bit ZigBee symbol from the MAC layer is spread to a predefined 32-chip pseudorandom noise (PN) sequence at the transmitter. VarChip sends a POTP by substituting it for some chips in the PN sequences. In addition, ZigBee adopts offset quadrature phase-shift keying (OQPSK) to (de)modulate the PN sequences outputted by DSSS. VarAmp embeds a POTP by increasing (or decreasing) the amplitude of an OQPSK symbol to convey a bit-1 (or bit-0). Finally, VarPhase embeds a POTP by manipulating the phase shifts between consecutive OQPSK signals according to predefined parameters. All three schemes only involve additional processing steps to existing PHY signal processing operations in ZigBee devices. In addition, they can be used independently or collectively as needed.

PhyAuth has many nice features that render a practical and effective defense against packet-injection attacks. (1) It is *invulnerable* to single point of compromise. Since POTPs use device-specific secret keys, the adversary can only use a

[1] AES-CCM stands for Advanced Encryption Standard-Counter with Cipher Block Chaining-Message Authentication Code.

compromised device itself to inject fake packets instead of impersonating other devices as in the case of message authentication based on the single network key. Continuous fake packets originating from the compromised device make it easily identifiable if the network administrator adopts fake-packet traceback defenses. (2) It is *standard-complaint*. The ZigBee specification defines the NWK and APL layers and assumes the use of IEEE 802.15.4 MAC and PHY layers. The actual PHY implementation (i.e., signal processing) is up to each device manufacturer as long as it offers proper services to the MAC layer. PhyAuth does not modify the ZigBee NWK/APL/MAC layers and only involves additional simple PHY processing logics. (3) It is *backward-compatible*. PhyAuth involves no hardware modification and can be implemented as a firmware update with slightly modified PHY signal processing logics. (4) It is *low-intrusive*. PhyAuth incurs a negligible negative impact on ZigBee PHY communication performance. ZigBee devices not implementing PhyAuth are oblivious to the POTPs embedded in PHY signals and can receive packets as usual. (5) It is *resilient* to fake and replayed POTPs. (6) It is *computationally efficient* due to using a standard hash function for POTP generation and verification.

We prototype PhyAuth on Universal Software Radio Peripheral (USRP) devices and thoroughly evaluate its performance in three representative environments: a laboratory room, a hallway, and an apartment. Our results show that PhyAuth can extract and verify POTPs from legitimate packets with an average false-negative rate of 0.8%. In addition, PhyAuth is highly resilient to fake packets with forged or replayed POTPs with an average false-positive rate of 0.01%. Moreover, PhyAuth has a negligible negative impact on normal ZigBee data transmissions.

2 ZigBee Basics

2.1 ZigBee Security 101

The ZigBee security architecture extends the basic security services provided by the underlying IEEE 802.15.4. It assumes an "open trust" model such that the protocol stack layers trust each other and that the layer that originates a frame is responsible for initially securing it. ZigBee communications are secured with 128-bit keys used with symmetric-key cryptographic building blocks including AES-CCM (an authenticated encryption algorithm) and AES-MMO (the Matyas-Meyer-Oseas hash function based on AES-128) [3].

ZigBee uses an entity known as the Trust Center to authenticate joining devices, distribute keys, and manage security policies. ZigBee supports two security models. There is exactly one active Trust Center (typically the ZigBee coordinator) in the centralized security model, while all ZigBee routers can act as the Trust Center in the distributed security model. The decision to use a centralized or distributed security model is made when the network is formed and cannot be changed afterward. Our PhyAuth can support both security models.

Broadcast and all NWK communications are secured with a 128-bit common network key in both distributed and centralized security models. Each device obtains the network key from the Trust Center in a secure fashion when admitted into the network. In addition, a ZigBee device is considered legitimate by its neighbors as long as it can send correctly formed NWK frames secured with the active network key.

End-to-end ZigBee communications are secured with a 128-bit *link key*. In particular, unicast communications between two APL peer entities are secured by a 128-bit *application link key* uniquely shared by the two devices, neither of which is the Trust Center. An application link key can be manually configured or established through the Trust Center which generates a key and sends it securely to two requesting devices. Each ZigBee device also maintains a 128-bit *Trust Center link key* for securing APS messages with the Trust Center which can be either global or unique for each device. Trust Center link keys may also be negotiated at the APL layer with a Certificate-Based Key Exchange protocol [3]. Fig. 2 shows the IEEE 802.15.4/ZigBee frame structure with all possible security features. MAC security can be optionally used based on the ZigBee keys. Both NWK and APS security can use only encryption, only authentication, or both. The MIC length can be 32, 64, or 128 bits in both NWK and APS frames.

2.2 ZigBee PHY Operations

ZigBee Transmitter. Fig. 16(a) shows how a ZigBee device transmits RF signals. Specifically, ZigBee first employs Direct Sequence Spread Spectrum (DSSS) to spread the bitstream from the MAC layer. Each byte of the bitstream is divided into two 4-bit ZigBee symbols with each mapped to a specified 32-chip PN sequence which is further modulated using OQPSK with half-sine pulse shaping. In particular, the odd and even chips are modulated as the in-phase (I) and quadrature (Q) components of the carrier wave, respectively. Both in-phase and quadrature chip sequences go through a half-sine pulse shaping module to shape the chips to a sinusoidal wave. Particularly, a chip-1 (or chip-0) is shaped to a positive (or negative) half-sine. Additionally, the quadrature chip sequence has a half-chip delay. Finally, the in-phase and quadrature signals are combined and transmitted to the air.

ZigBee Receiver. Fig. 16(b) shows the workflow of a ZigBee receiver. After receiving RF signals, the ZigBee receiver uses an Analog-to-digital converter (ADC) to digitize them into I/Q samples. Next, the ZigBee receiver uses the phase shift between consecutive I/Q samples to demodulate ZigBee symbols. Specifically, the phase shifts between consecutive I/Q samples are computed from $\arctan(Z(n) * Z^*(n-1))$, where $Z^*(n-1)$ is the conjugate of $Z(n-1)$. ZigBee outputs a chip-1 if the phase shift is bigger than 0° and otherwise a chip-0. After collecting a sequence of chips, ZigBee converts every 32-chip PN sequence to a 4-bit ZigBee symbol. However, due to noise and interference, some chips could be corrupted during transmission, leading to 32-chip PN sequences that do not match any of the 16 valid sequences. So ZigBee selects

the closest symbol with the smallest Hamming distance. In addition, users can define a correlation threshold to control the maximum Hamming distance between the received and the predefined 32-chip sequences that the ZigBee receiver can tolerate. Finally, the ZigBee receiver passes the decoded packets to the MAC layer.

3 PhyAuth Design

In this section, we first outline the PhyAuth workflow. Then we illustrate the generation and verification of POTPs. Finally, we present three schemes for embedding, transmitting, and extracting POTPs at the PHY layer.

3.1 PhyAuth Workflow

PhyAuth consists of two steps. (1) In the *sending step*, every legitimate ZigBee transmitter initiating or forwarding a packet must embed a POTP into the PHY signals of the preamble sequence in the 802.15.4/ZigBee frame (Fig. 2). It is worth noting that no change is made to the frame structure and standard frame processing at MAC/NWK/APS layers. (2) In the *receiving step*, every ZigBee receiver hearing the packet acts as a POTP verifier to extract and verify the POTP from the PHY signals of the preamble sequence. Only the packet with a valid POTP is passed to MAC/NWK/APS layers for routine ZigBee processing. The ZigBee receiver simply drops the packet without a valid POTP and can optionally notify the network administrator which can take further actions such as locating and excluding the network intruder. Since preamble processing is done at every ZigBee receiver regardless of whether it is the intended receiver, PhyAuth incurs negligible additional computational overhead for a simple bit-wise POTP comparison and an optional real-time hash operation.

3.2 POTP Generation and Verification

A POTP refers to a cryptographic and unforgeable binary sequence used for authorizing ZigBee devices to transmit packets within a ZigBee network. PhyAuth uses the following process for constructing and verifying POTPs.

POTP Generation. PhyAuth uses standard ZigBee security keys to construct POTPs by combining the HMAC-based OTP (HOTP) algorithm defined in RFC 4226 [1] and the Time-based OTP (TOTP) algorithm defined in RFC 6238 [2]. HOTP and TOTP are widely used in commercial two-factor authentication systems such as Google Authenticator [5] and Duo [4]. Both HOTP and TOTP use a secret key K known only to the HOTP/TOTP generator and verifier. The HOTP value is generated as $\mathsf{HOTP}(K,T) = \mathsf{Truncate}(\mathsf{HMAC}(K,C))$, where C denotes an increasing 8-byte counter, and $\mathsf{Truncate}$ represents the function that converts an HMAC value into a HOTP value as defined in [1]. In contrast, the TOTP value is computed as $\mathsf{TOTP}(K,T) = \mathsf{Truncate}(\mathsf{HMAC}(K,T))$ by replacing the counter C with the time factor T derived from a time reference and a time step.

The ZigBee transmitter generates the POTP value as

$$\mathsf{POTP}(K,T,\mathrm{SN},\text{src-addr}) = \mathsf{Truncate}(\mathsf{HMAC}(K,T,\mathrm{SN},\text{src-addr}))\,, \tag{1}$$

where K denotes a standard ZigBee security key; T is derived from a time reference and a time step; SN denotes the 8-bit monotonically increasing sequence number in the 802.15.4 MAC frame header; src-addr is the transmitter's 64-bit MAC address. The (T, SN, src-addr) triple serves as a time-based device-specific counter value that makes the resulting POTP both device-dependent and time-dependent with the desired one-time property. $\mathsf{HMAC}(\cdot)$ can use any cryptographic hash function implemented in software or hardware, such as AES-MMO (the standard ZigBee hash function [3]) or SHA-2 available on many commercial ZigBee hardware including CC1352P [8], CC2652P [9], and CC2652R [10]).

POTP Length. The HOTP/TOTP value should be at least 31-bit long for sufficient resilience to brute-force attacks [1,2]. So we require the POTP length to be at least 31 bits as well, e.g., 32/64/128 bits to match the security strength of MICs at NWK/APS layers (Fig. 2). The longer the POTP, the higher the attack resilience, the larger the related overhead, and vice versa. Such security-overhead trade-offs are analyzed in §4.

Time-step Size. The time factor T is an integer and represents the number of time steps between the initial counter time T_0 and the current Unix time. More specifically, let X represent the time step in seconds and T_0 denote the Unix time to start counting time steps. Both X and T_0 are system parameters and must be securely conveyed to each ZigBee device and POTP verifier when they join the system. We have $T = (\text{current Unix time} - T_0)/X$, where the default floor function is used in the computation. The time-step size X is set to less than the minimum time taken to wrap around the 8-bit MAC frame sequence number, which can be estimated as a common network parameter per the concrete ZigBee application and shared with all ZigBee devices.

Choice of Secret Key K. We use standard ZigBee security keys for the secret key K to generate the POTP and face three choices. As the first choice, K can be the common ZigBee network key, in which case a POTP can be verified by every ZigBee device and also dedicated verifiers that know the network key by default. Although simple, the exposure of the network key enables the adversary to derive a valid POTP for any illegitimate ZigBee device. As the second choice, K can be the transmitter's unique Trust Center link key which is known to the Trust Center and can be loaded to dedicated verifiers not engaged in ZigBee network operations. This option provides higher attack resilience because a compromised Trust Center link key allows the adversary to successfully impersonate the corresponding device only.

We opt for the last choice by setting K to an application link key shared between the transmitter and its neighbor(s). In particular, the ZigBee NWK layer has a neighbor-discovery

process that allows each device to discover and record one-hop neighbors for routing. We propose to use the standard ZigBee procedure (§2.1) to establish a unique application link key between any two neighboring ZigBee devices, as well as a common device-specific application link key each device shares with all its neighbors. Consider an arbitrary ZigBee transmitter i which has a common application link key k_i known to all its neighbors and a unique application link key $k_{i,j}$ with its neighbor j. Transmitter i uses k_i as K to generate the POTPs for broadcast packets, which can be verified by all its neighbors; it uses $k_{i,j}$ to derive the POTPs for unicast packets destined for device j, which can only be verified by device j. Since the Trust Center helps generates all application link keys in the ZigBee network, it can securely distribute all of them to dedicated POTP verifiers for them to validate all the POTPs.

POTP Verification. Let δ denote the integer-valued maximum possible clock drift normalized by the time-step size X between two ZigBee devices. Also assume that the current time-step window is T_c. Each POTP verifier extracts a POTP from the PHY signals of each incoming packet's preamble sequence and then verifies it in two steps for each time factor $T \in [T_c - \delta, T_c + \delta]$. (1) Check if the MAC frame sequence number is the largest it has ever seen from this transmitter in the time-step window T. (2) Use the broadcast/unicast application key associated with this transmitter as K to derive POTP(K,T, SN, src-addr) and check its equality with the extracted one. If either step fails, the POTP is considered invalid for the time-step window T. If the two-step verification fails for all $T \in [T_c - \delta, T_c + \delta]$, the verifier considers the transmitter illegitimate, drops the packet, and optionally reports this event to the network administrator. Otherwise, it passes the packet to the upper layers for routine ZigBee processing. The POTP verification involves a processing delay mainly incurred by at most $2\delta + 1$ hash operations with each to compute one anticipated POTP. If this already small real-time delay is a concern, each verifier can periodically precompute and store the POTPs for some future time-step windows. In this case, the real-time processing delay is reduced to the negligible time for POTP-table lookup and bit-wise POTP comparison.

3.3 POTP Encoding and Decoding

In this section, we illustrate three schemes to embed, transmit, and extract a POTP from PHY signals of an IEEE 802.15.4/ZigBee frame as shown in Fig. 2. All three schemes can embed a POTP to the PHY signals of the entire PHY frame as needed. For simplicity, we just insert the POTP into the preamble sequence as an example in what follows.

3.3.1 VarChip

POTP Embedding and Transmission. As mentioned in § 2.2, IEEE 802.15.4 uses DSSS to improve interference and noise resilience. In particular, ZigBee transmitters map every 4-bit ZigBee symbol to a 32-chip PN sequence. Since interference and noise can corrupt the chip stream during transmission, the received 32-chip sequences may not exactly match one of the 16 predefined standard PN sequences. As a result, a ZigBee receiver compares each received 32-chip PN sequence with the 16 predefined PN sequences and selects the corresponding symbol with the minimum Hamming distance. More importantly, DSSS defines a correlation threshold (e.g., 12) that can control the maximum Hamming distance between the received and predefined 32-chip PN sequences, allowing a tolerance margin for noise and interference resilience. We propose VarChip that uses the toleration margin to embed a POTP. Specifically, we replace some chips in a PN sequence by POTP bits. For simplicity, we illustrate VarChip by embedding POTP bits only into the PN sequence of the 4-byte PHY preamble, which corresponds to 8 symbols$\times$32 chips/symbol $= 256$ chips.

The key issue in VarChip is to determine the number of the embedded POTP bits and their embedding positions in the PN sequence of a PHY preamble. Our goal is to embed as many POTP bits as possible into one PHY packet while maintaining a very low symbol-error rate at the receiver. Suppose that we embed m_c bits into the 32-chip PN sequence of one symbol at the transmitter. Let S_t denote the set of chips in the modified 32-bit PN sequence that differ from those in the original 32-bit PN sequence. Since there may exist some common bits between the POTP bits and the chips that we want to replace in the original 32-bit PN sequence, we have $|S_t| \leq m_c$. In this paper, we consider the worst case, i.e., $|S_t| = m_c$. This means that we need to change m_c chips in the original 32-bit PN sequence. In addition, there may be some corrupted chips caused by interference and noise during transmission, which we denote by S_c. At the receiver, the hamming distance between the received and the corresponding 32-chip PN sequences is d_r with the following four cases: vspace-.1in

i $S_t \cap S_c = \emptyset$. None of the embedded POTP bits are corrupted during transmission. So we have $d_r = |S_t| + |S_c|$.

ii $S_t \cap S_c \neq \emptyset$. Not only some embedded POTP bits but also some original chips are corrupted during transmission. So we have $d_r = |S_t| + |S_c| - |S_t \cap S_c|$.

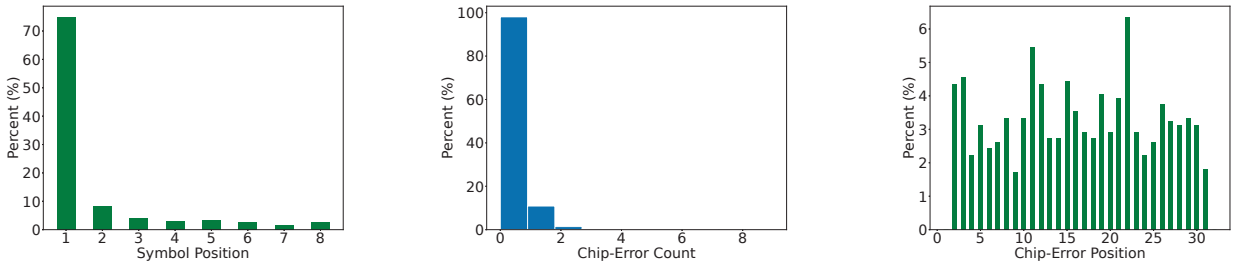
iii $S_t \subseteq S_c$. All the embedded POTP bits are corrupted during transmission. So we have $d_r = |S_c| - |S_t|$.

iv $S_c \subseteq S_t$. Only some embedded POTP bits are corrupted. So we have $d_r = |S_t| - |S_c|$.

Let θ denote the correlation threshold such that d_r must be $\leq \theta$ to ensure the correct decoding of the 32-chip PN sequence at the receiver. We have the following three cases:

i $S_t \cap S_c = \emptyset \rightarrow m_c = |S_t| \leq \theta - |S_c|$.

ii $S_t \cap S_c \neq \emptyset \rightarrow m_c = |S_t| \leq \theta - |S_c| + |S_t \cap S_c|$.



(a) Chip-error distribution over symbols (b) Distribution of chip-error count per symbol (c) Distribution of chip-error position

Figure 3: Chip-error pattern exemplification.

iii $S_t \subseteq S_c$ or $S_t \supseteq S_c \rightarrow |S_c| - \theta \leq m_c = |S_t| \leq \theta + |S_c|$.

The maximum value of m_c in the latter two cases is greater than that of m_c in the first case, indicating that we can embed more POTP bits if we replace some or all chips that expect to be corrupted during transmission. However, due to the randomness of the wireless communication channel, it is almost impossible to predetermine which chips will be corrupted during transmission. Consequently, we focus on the first case to determine the number of POTP bits that we can insert into a 32-chip PN sequence. Furthermore, the first case indicates that we should replace chips that cannot be corrupted during transmission by POTP bits, but each chip in the PN sequence may be corrupted in practice. Therefore, a more feasible way is to substitute POTP bits for chips with a low error probability. So how can a ZigBee transmitter identify the chip positions with a low error probability in a PN sequence?

We propose to let each ZigBee transmitter periodically estimate chip-error patterns from existing ZigBee packets. For example, 802.15.4/ZigBee uses periodic broadcast beacons to synchronize the network devices, which can be explored for our purpose. We use a pair of USRP B210s as the transmitter and receiver to illustrate the estimation process. The transmitter-receiver distance is set to 1 m, and the SNR is 18 dB. The transmitter sends 10 packets (e.g., beacons or data packets) within 10 ms to the receiver. So the receiver can get $256 \times 10 = 2560$ preamble chips to calculate the chip-error distribution. Fig. 3(a) shows the distribution of ZigBee symbol positions with chip errors. We can see that about 80% of chip errors occur in the first symbol of a PHY preamble. So we can embed POTP bits from the 2nd 32-chip PN sequence of the preamble. We further check the distribution of the chip-error count from symbol-2 to symbol-8. Fig. 3(b) shows that (1) about 89.3% symbols have no chip errors, and (2) about 99% symbols have 2 or fewer chip errors. So we can safely say that the number of chip errors for symbol-2 to symbol-8 lies in $[0,2]$. As a result, we can insert at most $7 \times (\theta - 2)$ bits into the 2nd to 8th 32-chip PN sequences of the preamble. For example, if θ is 10, we can embed at most $7 \times 8 = 56$ POTP bits into the PN sequence of the preamble. To determine the embedding positions, we check the distribution of chip-error positions from symbol-2 to symbol-8. Fig. 3(c) shows that without considering the 1st and 32nd chips, the chip positions with the lowest eight chip-error probabilities from low to high are the 9th, 31st, 4th, 24th, 6th, 7th, 25th, and 18th. Thus, we can choose these eight positions as the embedding positions for symbol-2 to symbol-8.

Based on the periodically estimated chip-error pattern, the transmitter embeds m_c POTP bits into a ZigBee symbol. Let N_p denote the number of preamble symbols available for POTP embedding. So the transmitter can embed an $m_c N_p$-bit POTP into the preamble of the PHY packet.

POTP Extraction. As a one-hop neighbor of the transmitter, each verifier can estimate the same or highly similar chip-error pattern. It then extracts the chips at the embedding positions to construct a candidate POTP for verification.

3.3.2 VarAmp

POTP Embedding and Transmission. As mentioned in § 2.2, ZigBee adopts OQPSK with half-sine pulse shaping to modulate the chip sequence outputted by the DSSS module. In particular, the chip sequence is first split into odd and even chip sequences, which are then assigned to the I and Q components of the carrier wave, respectively. Specifically, a chip-1 is encoded as a positive half sinusoid, while a chip-0 is encoded as a negative half sinusoid. Afterward, the OQPSK modulator sums the I component with the Q component delayed by a half-chip duration. This offset can limit the phase shift to no more than $\pm\pi/2$ at a time. OQPSK encodes two bits per symbol by using four phases: $\pi/4, 3\pi/4, 5\pi/4$, and $7\pi/4$, corresponding to four constellation points equispaced around a circle. We assume that the original four OQPSK constellation points (symbols) have an amplitude $\pm\sqrt{E}$.

From the OQPSK modulation process, we can see that the amplitudes of OQPSK symbols do not carry any information bits. Therefore, we can leverage the amplitudes of OQPSK symbols to convey POTP bits. We propose VarAmp to embed a POTP by dynamically changing the amplitudes of the original OQPSK symbols. Specifically, if the POTP bit is 1, we increase the amplitude of the original OQPSK symbol by a factor of α ($\alpha \geq 1$); if the POTP bit is 0, we decrease the am-

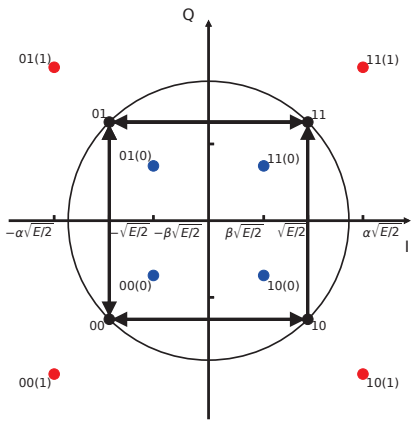


Figure 4: Constellation diagram for VarAmp.

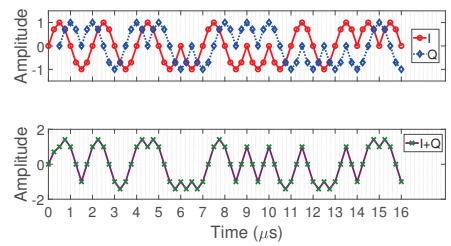


Figure 5: Half-sine wave for a zero-symbol chip sequence.

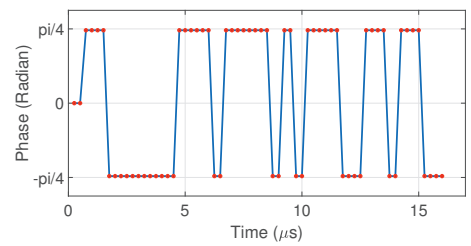


Figure 6: Phase shift for a zero-symbol.

plitude of the original OQPSK symbol by multiplying a factor of β ($0 < \beta < 1$). Here α and β here are system parameters whose impact is analyzed in § 5.

Fig. 4 shows the constellation diagram of VarAmp. The four black points represent the original constellation points $(\pm\sqrt{E/2}, \pm\sqrt{E/2})$. The phase shift between two adjacent constellation points is limited to $\pm\pi/2$, and the amplitude of each constellation point is $\sqrt{E}$ or $-\sqrt{E}$. In addition, the blue dots at $(\pm\beta\sqrt{E/2}, \pm\beta\sqrt{E/2})$ and red dots at $(\pm\alpha\sqrt{E/2}, \pm\alpha\sqrt{E/2})$ are POTP-constellation points. The bit value in parentheses represents the POTP bit. In addition, the two POTP-constellation points in each quadrant correspond to the same data symbol but different POTP bits. For example, the original constellation point for the OQPSK symbol "11" is $(\sqrt{E/2}, \sqrt{E/2})$. The transmitter sends $(\beta\sqrt{E/2}, \beta\sqrt{E/2})$ for a POTP bit-0 and $(\alpha\sqrt{E/2}, \alpha\sqrt{E/2})$ for a POTP bit-1.

We have two remarks to make. First, the larger α can induce higher transmission power. So α cannot be too large in practice due to many constraints. For example, FCC often imposes an upper limit on the transmission power, and the transmitter may have low energy residue. Second, similar to VarChip, VarAmp can embed a POTP into the preamble and other fields of a PHY packet.

POTP Extraction. The verifier extracts a POTP according to the constellation diagram of VarAmp. In particular, the verifier decodes POTP bits by checking the distances between the received data symbols and POTP-constellation points (blue and red dots) in Fig. 4. The verifier determines the POTP-constellation point closest to each received symbol and then decodes the embedded POTP bit as either 1 or 0.

3.3.3 VarPhase

POTP Embedding and Transmission. ZigBee uses phase shift between consecutive I/Q samples to demodulate ZigBee symbols. Specifically, ZigBee calculates the phase shift between two consecutive complex samples $Z(n)$ and $Z(n-1)$ by using $\arctan(Z(n) * Z^*(n-1))$, where $Z^*(n-1)$ is the conjugate of $Z(n-1)$. ZigBee outputs a chip-1 if the phase shift is bigger than 0° and otherwise a chip-0.

Based on the fact that the decoding of the chip value only relies on the sign ($\pm$) of the phase shift, we propose VarPhase, where legitimate transmitters embed a POTP into PHY packets by manipulating the phase shift between consecutive I/Q data samples. Specifically, when the POTP bit is 0, the original phase shift $\Delta\phi_o$ between two consecutive I/Q samples is scaled by μ ($\mu \geq 1$); when the POTP bit is 1, the original phase shift $\Delta\phi_o$ is scaled by λ ($\lambda > \mu$).

We first check the original phase shift between consecutive I/Q samples in PHY signals. The top figure in Fig. 5 shows the original half-sine waveform of a zero-symbol 32-chip PN sequence. The time duration of the zero-symbol is 16 µs. Each chip time (i.e., 1 µs) contains four discrete sampling points. We then merge the I and Q waveforms and get the resulting signal shown in the bottom figure of Fig. 5. Based on the summation signal, we further calculate the original phase shift between every two consecutive I/Q samples. Fig. 6 shows the original phase-shift sequence. We can see that the original phase shift $\Delta\phi_o$ could be $\frac{\pi}{4}$ or $-\frac{\pi}{4}$. Suppose that the current I/Q sample is $(A\sin\phi_o, A\cos\phi_o)$. If the POTP bit is 1, the next I/Q sample is $(A\cos(\phi_o + \lambda\Delta\phi_o), A\sin(\phi_o + \lambda\Delta\phi_o))$ and otherwise is $(A\cos(\phi_o + \mu\Delta\phi_o), A\sin(\phi_o + \mu\Delta\phi_o))$. Similar to VarChip and VarAmp, VarPhase can embed a POTP into the preamble and other fields of a PHY packet as needed.

POTP Extraction. To extract a valid POTP, the verifier de-

codes POTP bits by checking the phase shift between the received consecutive complex numbers. Specifically, if the phase shift is larger (smaller) than an empirical threshold τ, it decodes the embedded POTP bit as 1 (or 0).

4 Performance and Security Analysis

In this section, we theoretically analyze the POTP-decoding performance, communication and communication overhead, and security of VarChip, VarAmp, and VarPhase.

4.1 POTP-decoding Performance

We first analyze the bit error rate (BER) for the POTP. To make the analysis tractable, the channel is assumed to be Additive White Gaussian Noise (AWGN) with mean zero and power spectral density (PSD) $N_0/2$. We use E to represent the energy of an original constellation point. We define SNR as $\gamma = E/N_0$. According to [19], the original BER for the OQPSK modulation is $\mathsf{erfc}(\sqrt{\gamma/2})/2$, where $\mathsf{erfc}(\cdot)$ denotes the complementary error function. Given the encoding process in VarChip, its POTP BER at the verifier is the same as the BER in the original OQPSK. Therefore, we only show the following results for VarAmp and VarPhase.

Theorem 1. *The POTP BER of VarAmp is*

$$P_b^{AM} \approx \frac{\mathit{erfc}((\alpha-\beta)\sqrt{\gamma}/2)}{2}. \tag{2}$$

Proof. According to the nearest neighbor approximation [19], the BER is approximated as $\frac{M_{d_{\min}}}{2}\mathsf{erfc}(\frac{d_{\min}}{2\sqrt{N_0}})$, where $d_{\min}$ is the minimum distance between any two constellation points, and $M_{d_{\min}}$ is the number of neighbors separated by $d_{\min}$. In VarAmp, $d_{\min}$ equals $(\alpha-\beta)\sqrt{E}$, and $M_{d_{\min}}$ equals 1. So we can obtain Eq. (2). □

Theorem 2. *The POTP BER of VarPhase is*

$$P_b^{PH} \approx \mathit{erfc}(\sqrt{\gamma}\sin(\frac{\mu\Delta\phi_o}{2})). \tag{3}$$

Proof. In VarAmp, $d_{\min}$ is $2\sqrt{E}\sin(\mu\Delta\phi_o/2)$, and $M_{d_{\min}}$ is 2. We thus obtain Eq. (3). □

Next, we analyze the data BER of the PHY field (e.g., the preamble sequence) carrying POTP bits at the receiver. Ideally speaking, we would like PhyAuth to induce negligible changes in the data BER. Similar to the POTP BER, the data BER of VarChip is the same as that in the original QPSK constellation. We thus only show the following results for VarAmp and VarPhase.

Theorem 3. *For VarAmp, the data BER is upper-bounded by*

$$\overline{P_{b,data}^{AM}} \approx \frac{\mathit{erfc}(\beta\sqrt{\gamma/2})}{2}, \tag{4}$$

and lower-bounded by

$$\underline{P_{b,data}^{AM}} \approx \frac{\mathit{erfc}(\alpha\sqrt{\gamma/2})}{2}, \tag{5}$$

Proof. The larger amplitudes of the data symbols imply a higher SNR, leading to more error-resilient data transmission. Therefore, the upper bound of the data BER can be achieved when the POTP bits are all 0s so that the absolute amplitude of all data symbols is $\beta\sqrt{\gamma/2}$. So we can have Eq. (4). In contrast, the data BER can be minimized when the POTP bits are all 1s, so the absolute amplitude of all data symbols is $\alpha\sqrt{\gamma/2}$. We thus obtain Eq. (5). □

Theorem 4. *The data BER of VarPhase is upper-bounded by*

$$\overline{P_{b,data}^{PH}} \approx \mathit{erfc}(\sqrt{\gamma}\sin(\frac{\mu\Delta\phi_o}{2})), \tag{6}$$

and lower-bounded by

$$\underline{P_{b,data}^{PH}} \approx \mathit{erfc}(\sqrt{\gamma}\sin(\frac{\lambda\Delta\phi_o}{2})), \tag{7}$$

Proof. According to the nearest neighbor approximation, the greater distance between two constellation points implies a lower BER. Therefore, the upper bound of the data BER can be derived if the distance between the OQPSK constellation points is $2\sqrt{E}\sin(\mu\Delta\phi_o/2)$. So we can have Eq. (6). Correspondingly, the lower bound is achieved when the distances between the OQPSK constellation points are always increased. In this case, the mutual distance between the QPSK constellation points is $2\sqrt{E}\sin(\lambda\Delta\phi_o/2)$. We thus obtain Eq. (7). □

Let p denote the POTP BER of VarChip/VarAmp/VarPhase as derived above. The probability for each verifier to correctly decode an N_l-bit POTP is simply $(1-p)^{N_l}$, which can be further improved by using error-correction codes such as the Reed-Solomon code in IEEE 802.15.4 to encode the POTP.

4.2 Communication/Computation Overhead and Energy Consumption

PhyAuth does not introduce extra ZigBee traffic except the one-time packets for establishing application link keys, so its runtime communication overhead is negligible.

The computation overhead and energy consumption of PhyAuth are dominated by the $\mathsf{HMAC}(\cdot)$ operation in Eq. (1) for generating/verifying a POTP. Each HMAC operation involves two passes of a cryptographic hash function such as SHA-1/SHA-2/AES-MMO. The HMAC input is 128-bit long, corresponding to the bit-wise XOR on a 128-bit application link key, a 32-bit time factor T, a 64-bit MAC address, and a 8-bit frame sequence number. Each intermediate ZigBee node (i.e., router) performs one HMAC operation to verify the POTP in the incoming packet and the other to generate a new POTP in the outgoing packet to the next hop. In contrast, each intermediate node uses the 128-bit network key to perform

two AES-CCM operations on an input up to 100 bytes long in the current ZigBee security architecture, one for verifying the NWK MIC in each incoming packet and the other for regenerating it for the outgoing packet.

We use the benchmark results in the TI report [6] to compare PhyAuth with the original ZigBee hop-by-hop message authentication method. Assume that the input length of AES-CCM and SHA-224 (an SHA-2 hash function) is 64 bytes. For hardware implementations of AES-CCM and SHA-224 in the popular SimpleLink CC13x2/CC26x2 ZigBee devices, the duration of an AES-CCM operation with a 128-bit key is 0.041 ms with the average current of 3.9mA, while the duration of an SHA-224 operation is 0.024 ms with the average current of 3.8mA. For Arm Cortex-M4F software-based implementations of AES-CCM and SHA-224, the duration of an AES-CCM operation with a 128-bit key is 0.435 ms with the average current of 3.1mA, while the duration of an SHA-224 operation is 0.179 ms with the average current of 3.1mA. Since each HMAC operations involves two SHA-224 operations, the energy consumption of each POTP generation/verification is (2*0.024*3.8)/(0.041*3.9)$\approx$ 1.1 times that of each NWK MIC verification/generation for hardware implementations and (2*0.179*3.1)/(0.435*3.1)$\approx$ 0.8 times for software implementations. The actual POTP involves an 128-bit input, while the real NWK MIC can involve an input up to 100 bytes long. It is safe to say that the computational delay and energy consumption of the POTP generation/verification in PhyAuth are at least as good as the NWK-layer MIC generation/verification in the ZigBee security architecture. PhyAuth does add extra overhead for per-hop POTP verification/generation involving legitimate ZigBee packets, which can be easily mitigated. In particular, the network administrator can turn off the NWK-layer MIC operation by setting the proper option bits in the auxiliary NWK header when PhyAuth is always activated or on demand, e.g., when there is evidence of huge fake traffic.

The huge energy savings of PhyAuth lie in its capability to stop the multi-hop or network-wide transmissions of fake packets. Without PhyAuth in place, a fake unicast packet sent over a n-hop path involves $2n-1$ AES-CCM operations (two by each intermediate node and one by the destination), and a fake broadcast packet to a network of N nodes involves up to $2N$ AES-CCM operations (two by each node). With PhyAuth in place, a fake unicast/broadcast packet with an incorrect POTP but a correct NWK MIC can be immediately detected and dropped by the legitimate neighbors of the attacker device after each performs one POTP verification.

4.3 Security Analysis

We assume that attackers can use commodity software-defined radios like USRPs to send fake ZigBee packets with an arbitrary spoofed address. By continuously transmitting fake packets, attackers aim to disrupt network operations by consuming the network bandwidth as well as the processing power, memory, and battery of ZigBee devices. Attackers are also assumed to be computationally bounded and cannot break the cryptographic primitives used by PhyAuth. ZigBee and all other wireless networks are vulnerable to naive jamming attacks that use random signals to jam the entire frequency band. How to deal with such traditional jammers is beyond the scope of this paper.

We first consider *external* attackers that do not know the authentic network key of the target ZigBee network. Without PhyAuth in place, fake packets do not carry a correct NWK MIC and thus cannot pass the original hop-by-hop message authentication that involves an AES-CCM-based MIC verification. With PhyAuth in place, external attackers must insert a forged POTP into the PHY signals of each fake packet. As long as the POTP value is long enough (say, $L \geq 32$ bits per the HOTP security recommendation [1]), the probability $1/2^L$ that fake packets with a randomly forged POTP pass PhyAuth detection can be made very small. Since POTP verification can be computationally much more efficient than an AES-CCM-based MIC verification (§4.3), it is also computationally beneficial to filter fake packets with PhyAuth before invoking the NWK MIC verification.

We then consider *internal* attackers that have compromised an arbitrary node A to get the authentic network key. Without PhyAuth in place, an internal attacker can use the compromised network key to send fake packets in the name of any spoofed node, say B. Since such fake packets carry an authentic NWK MIC, they can pass hop-by-hop NWK-layer message authentication to reach the intended destinations. With PhyAuth in place, the internal attacker must send the correct POTP for each fake packet purportedly sent by B, which succeeds with a negligible probability $1/2^L$. Fake packets without an authentic POTP can be immediately detected and dropped by the legitimate one-hop neighbors of the attacker. Since impersonating arbitrary nodes to send fake packets is no longer feasible, internal attackers can only use the compromised node A to send fake packets which carry correct POTPs and also NWK MIC values. Although such fake packets can evade hop-by-hop message authentication by PhyAuth and also the ZigBee NWK layer, they make A easily identifiable if the network administrator employs simple fake-packet trackback mechanisms.

PhyAuth is highly resilient to the replay attack as well. In particular, each POTP value is dependent on the transmitter's MAC address, MAC frame sequence number, and the time factor, it satisfies the one-time property required of an OTP algorithm. In addition, since the security key for generating POTPs is the broadcast/unicast application link key each ZigBee transmitter shares with its authenticated one-hop neighbors, each POTP can only be used in the one-hop neighborhood of the corresponding transmitter. Therefore, both external and internal attackers cannot replay a sniffed POTP to impersonate the target transmitter in its vicinity or other areas of the ZigBee network. This also implies that PhyAuth

does not have a higher time synchronization requirement than the current ZigBee network to deal with the replay attack.

Finally, the adversary may compromise a neighboring node A of a target node B to acquire its broadcast/unicast application key to generate B's valid POTPs. But the adversary cannot use such valid POTPs to send fake packets without being detected, as these POTPs can only be used in B's one-hop neighborhood and thus can be detected by B itself.

5 Performance Evaluation

In this section, we thoroughly evaluate PhyAuth using USRP experiments. In what follows, we first describe our experimental setup. Then we evaluate the performance of the three schemes under different scenarios.

5.1 Experimental setup

We implement PhyAuth with three USRP devices as the hardware platforms. Specifically, we connect one N210 USRP to a Dell laptop with an Ethernet cable and use it as the ZigBee receiver. One B210 USRP serves as the ZigBee transmitter, and the other B210 USRP acts as the dedicated verifier. The two B210 USRPs are connected to two Dell laptops which have two Intel 4.7 GHz i7 processors where all the computations are executed. In addition, we implement the three PhyAuth schemes (i.e., VarChip, VarAmp, and VarPhase) on GNU Radio by modifying the open source code of IEEE 802.15.4 PHY [15]. We use the Nyquist sampling rate 4 MHz (i.e., twice the bandwidth). SHA-1 is used to implement the HMAC function in Eq. (1) to generate POTPs of 32 bits or longer with the time step X set to 30 s. We embed each POTP into randomly generated data packets with a constant payload length of 100 bytes. Additionally, we use three representative physical environments in the evaluations: (1) a laboratory room with the size of $8\,\text{m} \times 6\,\text{m}$ (Fig. 7(a)), (2) a hallway (Fig. 7(b)), and (3) an apartment (Fig. 7(c)).

In our experiments, we evaluate the impact of the distance and the channel SNR. Specifically, in the laboratory room and the hallway, the transmitter-receiver and transmitter-verifier distance settings include 1 m, 4 m, 7 m, and 10 m. The receiver-verifier distances are fixed to about 2 m. In the apartment, we deploy the receiver and the verifier in the kitchen, and the transmitter in bedroom 1 or 2. The transmitter-receiver and transmitter-verifier distances are about 9 m. In addition, at each position, we evaluate the three schemes with 5 different SNRs: 10 dB, 14 dB, 16 dB, 18 dB, 20 dB. The experienced channel noises are from the natural physical environment. For all experimental environments, there are random human activities such as walking during the experiments.

5.2 Performance Metrics

We use four performance metrics as follows. (1) The first is the POTP-bit error rate (PBER) defined as $\text{PBER} = \frac{N_e^B}{N_L^B}$, where N_e^B denotes the number of received POTP-bit errors, and N_L^B denotes the POTP length. We use PBER to evaluate the performance of the three schemes on decoding POTPs from the received data packets. (2) The second is the packet error rate (PER) defined as $\text{PER} = \frac{N_e^P}{N_r^P}$, where N_e^s and N_r^P denote the number of incorrectly received and total ZigBee packets, respectively. We use PER to evaluate the impact of the three schemes on normal data transmission/reception. (3) The third is the false-negative rate (FNR) defined as $\text{FNR} = \frac{N_e^{FN}}{N_r^P}$, where N_e^{FN} and N_r^P denote the number of legitimate packets that are incorrectly classified as fake packets and total received legitimate packets, respectively. (4) The fourth is the false-positive rate (FPR) defined as $\text{FPR} = \frac{N_e^{FP}}{N_r^P}$, where N_e^{FP} and N_r^P denote the number of fake packets that are incorrectly classified as legitimate packets, and total received fake packets, respectively. We use FNR and FPR to evaluate the ability of the three schemes to distinguish between legitimate and fake packets.

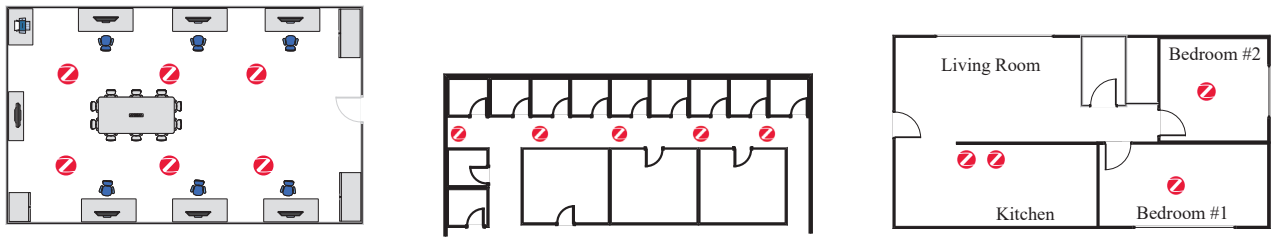
5.3 Parameter Selection

5.3.1 VarChip

In VarChip, we need to configure three parameters: N_p, m_c, and S_e. In our experiments, the transmitter sends 10 beacon packets within 10 ms in every time step to determine these three parameters. Our results show that the chip error patterns are different for different scenarios. We also observe that about 80% to 95% chip errors occur in the first preamble symbol. Therefore, we can embed POTP bits from the 2nd preamble symbol and set N_p to 7 in all experiments. For the 2nd to 8th preamble symbol, we further check the frequency distribution of chip-error count in each preamble symbol. Then we can find the chip-error count N_{ec} in the 97th percentile. This indicates 97% preamble symbols have N_{ec} or fewer chip errors. We then set m_c to $\theta - N_{ec}$, where θ is the correlation threshold and equals 10 in our experiments. Our experimental results show that m_c decreases as SNR decreases. This is reasonable because a channel with a lower SNR can corrupt more chips during transmission. After this, we check the chip-error position distribution to find m_c chip positions with the lowest error probability and put them in the embedding-position set S_e. In our experiments, we find that m_c and the chip-error position distributions change under different scenarios (e.g., different SNRs and time). The reason is that the wireless channel is not very stable in different scenarios. We thus can not list all the settings of the three parameters for all scenarios due to space limitations. It is also worth noting that m_c is no less than 5 in all experiments, indicating that we can insert at least $5 \times 7 = 35$ POTP bits into one packet with VarChip.

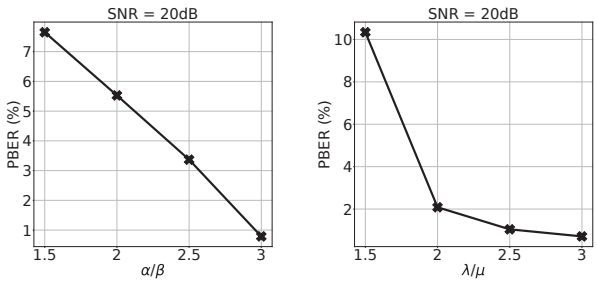
5.3.2 VarAmp

In VarAmp, we embed a 128-bit POTP into one PHY packet. We also need to set up two parameters: α and β. We evaluate the impact of α and β on PER, LRR, and PBER at different positions with different SNRs. In our experiments, α is set



(a) Lab (b) Hallway (c) Apartment

Figure 7: The floor plan of the lab, hallway and apartment.



(a) α/β v.s. PBER (b) λ/μ v.s. PBER

Figure 8: Parameter settings in VarAmp and VarPhase.

to 1.2|1.4|1.6|1.8, β is set to 0.8|0.7|0.65|0.6. So the ratio α/β is 1.5|2|2.5|3. Fig. 8(a) shows the impact of α/β on PBER when the SNR is 20 dB. As expected, the PBERs dramatically decrease as α/β increases. We can also observe similar results under other scenarios. Since α/β relates to the distance between two POTP-constellation points in each quadrant, the larger α/β, the greater distance between two POTP-constellation points in each quadrant, the lower detection error for POTP bits, and versa. In addition, we evaluate the impact of α/β on PER and have similar observations. Based on these observations, we set $\alpha = 1.8$ and $\beta = 0.6$ in our experiments.

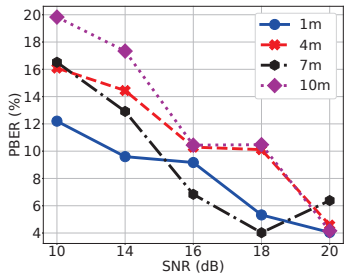
5.3.3 VarPhase

Similar to VarAmp, VarPhase can also embed a 128-bit POTP into one packet. In VarPhase, we need to determine three parameters: λ, μ, and τ. Similar to VarAmp, we evaluate the impact of λ and μ on PBER at different positions with different SNRs. In our experiments, λ is set to 1.5, 2, 2.5, 3, μ is set to 1 for all scenarios. So the ratio of λ to μ is 1.5, 2, 2.5, 3. Fig. 8(a) shows the impact of λ/μ on PBER when SNR is 20 dB. As expected, the PBERs decrease dramatically as λ/μ increases. Similar results can also be observed under other scenarios. Similar to VarAmp, a larger λ/μ indicates a greater distance between two consecutive constellation points, inducing a lower detection-error rate for POTP bits. We also evaluate the impact of λ/μ on PER and have similar observations. Therefore, we set λ and μ to 3 and 1, respectively. Additionally, we empirically set τ to 0.85 based on our experimental results.

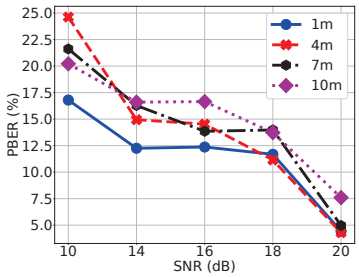
5.4 PhyAuth Performance

In our experiments, we let the ZigBee transmitter send 5,000 packets with POTPs embedded in their PHY preambles. Since VarAmp increases or decreases the amplitude of the original OQPSK symbols if a POTP bit is 1 or 0, the average transmission power consumption does not change if POTP bit-1 and bit-0 occur with equal probability. Fig. 9 shows the PBER performance for our three schemes under all scenarios. As expected, we can observe the PBER curves for all three schemes decreases as SNR increases from 10 dB to 20 dB. In addition, the four curves in each figure are close to each other, indicating that the distance impact on the three schemes is negligible. Moreover, compared with VarChip and VarAmp, VarPhase can achieve lower PBERs, which is consistent with the analysis in Eq. (2) and Eq. (3). Specifically, when the SNR is larger than 16 dB, the PBERs of VarPhase in all scenarios are below 10%. In addition, when the SNR is larger than 18 dB, the PBERs of the three schemes in almost all scenarios are below 10%. These results demonstrate that the three schemes can effectively extract and decode POTPs in practice. We place more statistical results about PBER in Appendix B. Moreover, we can encode the POTP with a lightweight error-correction code (ECC) to dramatically improve the detection performance. For example, we can simply repeat every POTP bit three times and using the majority vote for decoding. Fig. 15 shows the PBER performance for VarPhase at the hallway. Compared with Fig. 9(h), the PBERs can be reduced about by 50%. The error tolerance or detection performance can be further improved if more sophisticated ECCs such as the Reed-Solomon code are used to encode POTP bits.

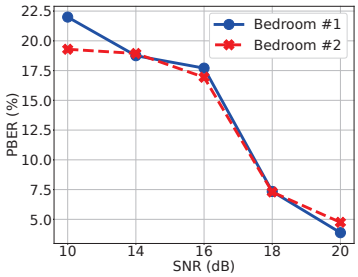
Fig. 10 illustrates the PER at the receiver of the three schemes. According to Eq. (4) to (7), both VarAmp and VarPhase can lower the BER, resulting in a reduction in PERs in contrast to the original OQPSK modulation. In addition, we notice that VarChip can increase the PERs in low SNR cases. The reason for this is that we replace some chips in the



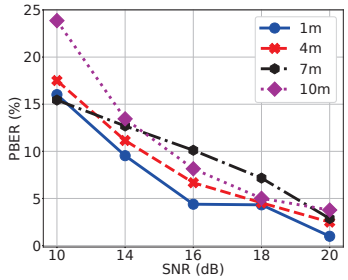
(a) VarChip at Lab



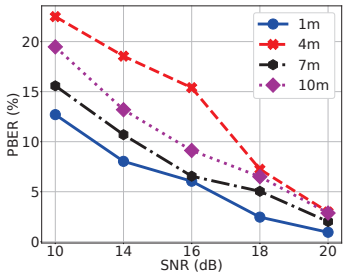
(b) VarChip at Hallway



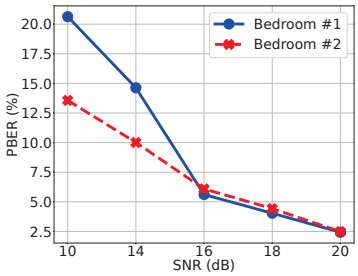
(c) VarChip at Apartment



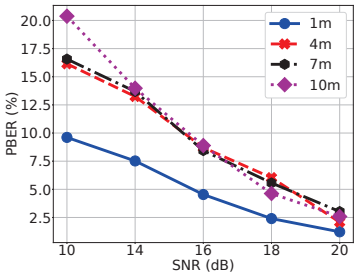
(d) VarAmp at Lab



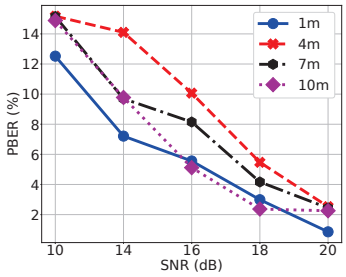
(e) VarAmp at Hallway



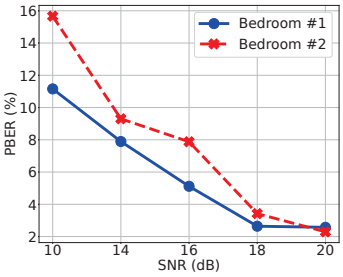
(f) VarAmp at Apartment



(g) VarPhase at Lab



(h) VarPhase at Hallway



(i) VarPhase at Apartment

Figure 9: PBER performance.

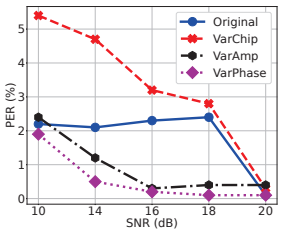


Figure 10: PER performance.

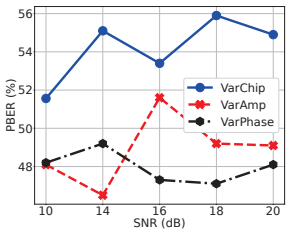


Figure 11: No POTP.

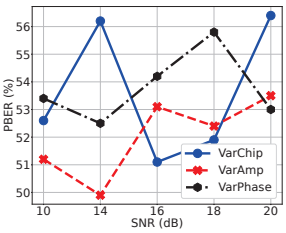


Figure 12: Fake POTPs.

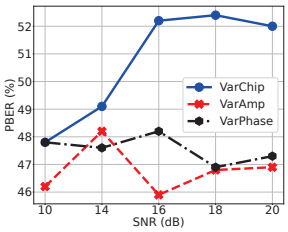


Figure 13: Replay POTPs.

original PN sequences with POTP bits, thus increasing the chip-error count in each packet. However, VarChip has almost no negative impact on the PER when the SNR is above 18 dB in normal communication environments.

Fig. 14 shows the average FNRs of the three schemes under different scenarios. In our experiments, we set a threshold to determine if a decoded POTP is valid or not. Specifically, if the PBER of a decoded POTP is below the predefined threshold, we consider the corresponding received packet legitimate and otherwise fake. In VarChip, VarAmp, and VarPhase, we set the threshold to 25%. We can see that the average FNRs decrease as the SNR increases. When the SNR is above 14 dB, the

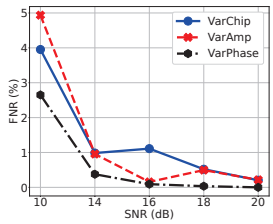


Figure 14: FNR performance.

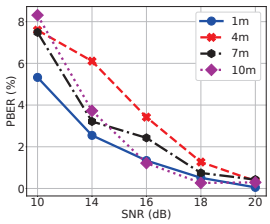


Figure 15: PBER (r = 3).

FNRs of the three schemes can be lower than 1%. These results demonstrate that the three schemes can extract and verify POTPs from legitimate packets with a very low FNR.

We also experimentally evaluate the attack resilience of our three schemes. In our experiments, we use one USRP B210 as the attacker to transmit 5,000 packets without any POTP, with fake POTPs, and with replayed valid POTPs. The verifier always follows the PhyAuth protocol and tries to detect and decode POTPs from each received packet regardless of the adversary's actions. We then calculate the average PBER from the extracted POTPs. Fig. 11, Fig. 12, and Fig. 13 show the average PBER performance of the three schemes under the three attacks. As we can see, the average PBERs of decoded POTPs extracted from fake packets in all schemes are above 45%, which is much greater than those of legitimate POTPs. Furthermore, we evaluate the FPRs of the three schemes with the same setting for FNRs. As expected, the average FPRs of VarChip, VarAmp, and VarPhase can be as low as 0.01%. These results show the high efficacy (i.e., attack resilience) of the three schemes for fake-packet detection.

To summarize, VarChip, VarAmp, and VarPhase have slightly different performance for various SNR contexts but are comparably very good overall. In practice, the network administrator can activate one of them based on onsite SNR and PBER measurements. Any combination of the three schemes can also be used to improve the error tolerance of POTP transmissions or even implement a PHY covert channel without violating the ZigBee specification.

6 Related Work

The most related work to PhyAuth is PHY authentication (PLA) in wireless networks. According to [31], we classify existing works into passive and active PLA schemes.

In the passive PLA schemes, a receiver uses the features of the received PHY signals to verify the transmitter. The scheme in [24] uses differential constellation trace figure (DCTF) to identify ZigBee devices. But the authors only evaluate their scheme on 16 ZigBee devices. The scheme in [25] uses DCTF, carrier frequency offset (CFO), modulation offset, and I/Q channel offset features extracted from constellation trace figure (CTF) to train a hybrid ML classifier to identify different ZigBee devices. Their results show that there is a 4%–9% loss of classification accuracy under multipath fading scenarios. The scheme in [34] extracts statistics from the preambles of ZigBee devices and uses the Mahalanobis distance and nearest neighbor algorithm to identify 50 ZigBee devices. In addition, the schemes in [23, 26, 32] employ DNN models to extract latent features from received signals to distinguish ZigBee devices. But it requires a large amount of data to train a DNN model. Moreover, their results show that the performance of the trained models suffers a great loss of degradation in accuracy for low SNR cases (e.g., SNR < 15 dB). Furthermore, these schemes cannot be deployed on resource-constrained ZigBee devices.

PhyAuth belongs to active PLA schemes, in which a transmitter constructs an authentication token based on a symmetric key and embeds it into a PHY packet. Then the intended receiver extracts and verifies the authentication token from the received packet. The schemes in [33] and [28] superimpose the authentication token to the original 16-QAM and QPSK signal. The schemes in [22] and [30] add the authentication token into the PHY packet by replacing some initialization bits in the original message with the corresponding token bits. In [18], Goergen *et al.* hide the authentication in the channel fading. Auth-SLO [29] divides the transmitting signals into two groups according to a pre-shared secret key and manipulates the transmission power of each group marginally. The scheme in [16] superimposes an authentication token to QAM signals by controlling the disturbance of the transmitted symbol around constellation points. The scheme in [20] embeds a spectrum bit into an OFDM frame by dynamically changing its cyclic prefix length. Unlike these existing works, our work focuses on ZigBee networks that have a different data modulation scheme and thus require different methods for embedding an authentication token into PHY signals.

7 Conclusion and Discussion

In this paper, we presented the design and evaluation of PhyAuth, a PHY message authentication framework against packet-inject attacks in ZigBee networks. ZigBee devices commonly adopt the wireless microcontrollers (MCUs) made by major manufacturers such as TI, NXP, Microchip, and Silicon Labs. To deploy PhyAuth in practice, these ZigBee MCU makers need to update the firmware to incorporate the required PHY signal processing logic and also provide the corresponding API for this additional security feature to be invoked by the application profile of the corresponding ZigBee device. PhyAuth does have a few limitations. First, it incurs additional PHY processing overhead for legitimate packets due to two HMAC operations for POTP verification and generation, respectively. This limitation can be mitigated if the network administrator only activates PhyAuth when a certain amount of fake traffic is reported, which requires the deployment of a network intrusion system. Second, some existing ZigBee devices may not easily support PHY firmware updates, in which case PhyAuth cannot apply.

Acknowledgement

We thank the anonymous reviewers and shepherd for their invaluable advice that have greatly helped improve the quality of this paper. This paper was supported in part by U.S. National Science Foundation under awards CNS-1652669, CNS-1824355, CNS-1824491, CNS-1933069, and CNS-2055751.

References

[1] HOTP: An HMAC-Based One-Time Password Algorithm. https://datatracker.ietf.org/doc/html/rfc4226, 2005.

[2] TOTP: Time-Based One-Time Password Algorithm. https://datatracker.ietf.org/doc/html/rfc6238, 2011.

[3] ZigBee Pro Specification. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf, 2015.

[4] Duo security. https://duo.com/, 2018.

[5] Google 2-Step Verification. https://safety.google/authentication/, 2018.

[6] Cryptographic Performance and Energy Efficiency on SimpleLink™ CC13x2/CC26x2 Wireless MCUs. https://www.ti.com/lit/an/swra667/swra667.pdf?ts=1662470814398, 2020.

[7] IEEE Standard for Low-Rate Wireless Networks. https://standards.ieee.org/ieee/802.15.4/7029/, 2020.

[8] CC1352P SimpleLink™ High-Performance Multi-Ban Wireless MCU With Integrated Power Amplifier. https://www.ti.com/lit/ds/symlink/cc1352p.pdf?ts=1662950426920&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC1352P%253FkeyMatch%253DCC1352%2526tisearch%253DSearch-EN-Everything, 2021.

[9] CC2652P SimpleLink™ High-Performance Multi-Ban Wireless MCU With Integrated Power Amplifier. https://www.ti.com/lit/ds/symlink/cc2652r.pdf?ts=1662950398077&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC2652R%253FkeyMatch%253DCC1352%2526tisearch%253DSearch-EN-Everything, 2021.

[10] CC2652R SimpleLink™ Multiprotocol 2.4 GHz Wireless MCU. https://www.ti.com/lit/ds/symlink/cc1352r.pdf?ts=1662914186089&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC1352R, 2021.

[11] Connected Solutions | Market Uses. https://csa-iot.org/market-uses/, 2021.

[12] Smart Factory Solutions with Zigbee. https://www.icpdas-usa.com/Smart-Factory-Solutions-with-Zigbee.html, 2021.

[13] Innovation & Adoption – Zigbee momentum in 2021. https://csa-iot.org/newsroom/innovation-adoption-zigbee-momentum-in-2021/, 2022.

[14] Zigbee FAQ. https://csa-iot.org/all-solutions/zigbee/zigbee-faq/, 2022.

[15] Bastian Bloessl, Christoph Leitner, Falko Dressler, and Christoph Sommer. A GNU Radio-based IEEE 802.15.4 Testbed. In *FGSN*, Cottbus, Germany, September 2013.

[16] Kapil Borle, Biao Chen, and Wenliang Du. Physical layer spectrum usage authentication in cognitive radio: Analysis and implementation. *IEEE Transactions on Information Forensics and Security*, 10(10):2225–2235, July 2015.

[17] ShuYu Ding, JianLi Liu, and MingHong Yue. The Use of ZigBee Wireless Communication Technology in Industrial Automation Control. *Wireless Communications and Mobile Computing*, 2021, December 2021.

[18] Nate Goergen, W Sabrina Lin, KJ Liu, and T Charles Clancy. Extrinsic channel-like fingerprinting overlays using subspace embedding. *IEEE Transactions on Information Forensics and Security*, 6(4):1355–1369, October 2011.

[19] A. Goldsmith. *Wireless communications*. Cambridge University Press, 2005.

[20] Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang. SafeDSA: Safeguard dynamic spectrum access against fake secondary users. In *ACM CCS*, Denver, CO, October 2015.

[21] M.Barathi Kannamma, B.Chanthini, and D.Manivannan. Controlling and monitoring process in industrial automation using Zigbee. In *IEEE ICACCI*, Mysore, India, August 2013.

[22] Vireshwar Kumar, Jung-Min Park, and Kaigui Bian. PHY-layer authentication using duobinary signaling for spectrum enforcement. *IEEE Transactions on Information Forensics and Security*, 11(5):1027–1038, January 2016.

[23] Kevin Merchant, Shauna Revay, George Stantchev, and Bryan Nousain. Deep learning for RF device fingerprinting in cognitive communication networks. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):160–167, January 2018.

[24] Linning Peng, Aiqun Hu, Yu Jiang, Yan Yan, and Changming Zhu. A differential constellation trace figure based device identification method for ZigBee nodes. In *IEEE WCSP*, Yangzhou, China, October 2016.

[25] Linning Peng, Aiqun Hu, Junqing Zhang, Yu Jiang, Jiabao Yu, and Yan Yan. Design of a hybrid RF fingerprint extraction and device classification scheme. *IEEE Internet of Things Journal*, 6(1):349–360, May 2018.

[26] Linning Peng, Junqing Zhang, Ming Liu, and Aiqun Hu. Deep learning based RF fingerprint identification using differential constellation trace figure. *IEEE Transactions on Vehicular Technology*, 69(1):1091–1095, October 2019.

[27] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, July 2003.

[28] Yachao Ran, Harith Al-Shwaily, Chaoqing Tang, Gui Yun Tian, and Martin Johnston. Physical layer authentication scheme with channel based tag padding sequence. *IET Communications*, 13(12):1776–1780, July 2019.

[29] Ning Xie and Changsheng Chen. Slope authentication at the physical layer. *IEEE Transactions on Information Forensics and Security*, 13(6):1579–1594, January 2018.

[30] Ning Xie, Changsheng Chen, and Zhong Ming. Security model of authentication at the physical layer and performance analysis over fading channels. *IEEE Transactions on Dependable and Secure Computing*, 18(1):253–268, November 2018.

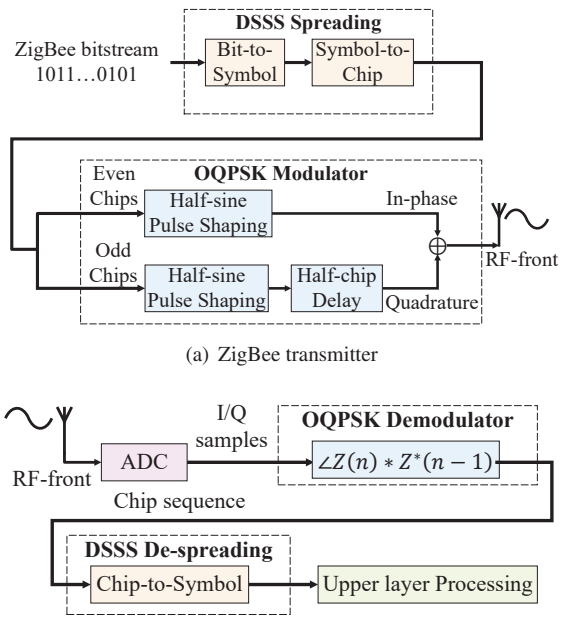
[31] Ning Xie, Zhuoyuan Li, and Haijun Tan. A survey of physical-layer authentication in wireless communications. *IEEE Communications Surveys & Tutorials*, 23(1):282–310, December 2020.

[32] Jiabao Yu, Aiqun Hu, Guyue Li, and Linning Peng. A robust RF fingerprinting approach using multisampling convolutional neural network. *IEEE Internet of Things Journal*, 6(4):6786–6799, April 2019.

[33] Pinchang Zhang, Jun Liu, Yulong Shen, Hewu Li, and Xiaohong Jiang. Lightweight tag-based PHY-layer authentication for iot devices in smart cities. *IEEE Internet of Things Journal*, 7(5):3977–3990, December 2019.

[34] Xinyu Zhou, Aiqun Hu, Guyue Li, Linning Peng, Yuexiu Xing, and Jiabao Yu. Design of a robust RF fingerprint generation and classification scheme for practical device identification. In *IEEE CNS*, Washington, DC, June 2019.

A ZigBee Communication Basics



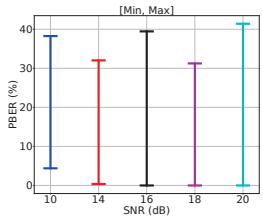
(a) ZigBee transmitter

(b) ZigBee receiver

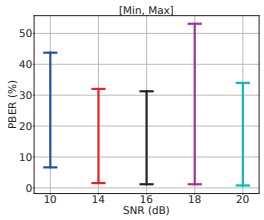
Figure 16: Illustrations of ZigBee transmitters and receivers.

B Statistical PBER Results

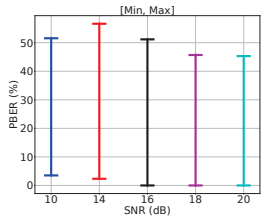
Fig. 17, Fig. 18 and Fig. 19 show the maximum, minimum and 95% confidence intervals of PBERs under different scenarios for VarChip, VarAmp, and VarPhase.



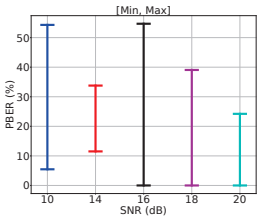
(a) VarChip at Lab (1 m)



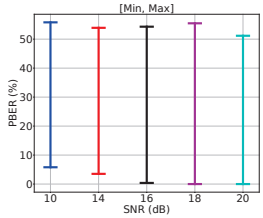
(b) VarChip at Lab (4 m)



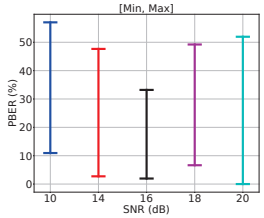
(c) VarChip at Lab (7 m)



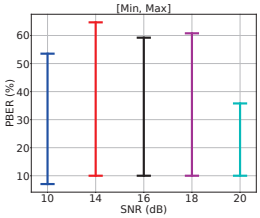
(d) VarChip at Lab (10 m)



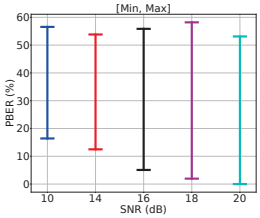
(e) VarChip at Hallway (1 m)



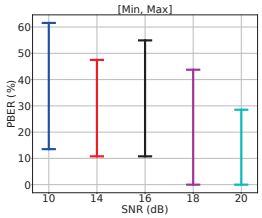
(f) VarChip at Hallway (4 m)



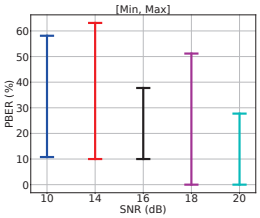
(g) VarChip at Hallway (7 m)



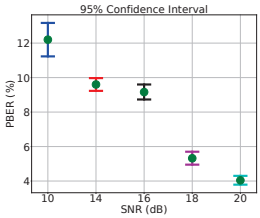
(h) VarChip at Hallway (10 m)



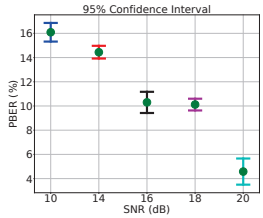
(i) VarChip at Room #1



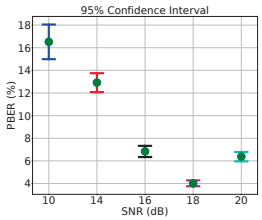
(j) VarChip at Room #2



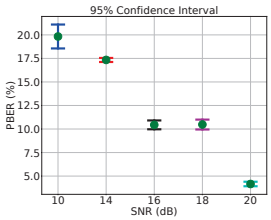
(k) VarChip at Lab (1 m)



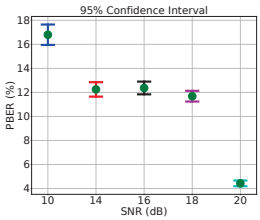
(l) VarChip at Lab (4 m)



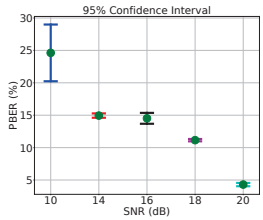
(m) VarChip at Lab (7 m)



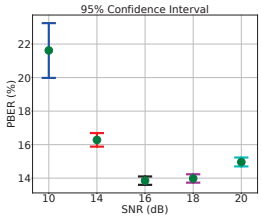
(n) VarChip at Lab (10 m)



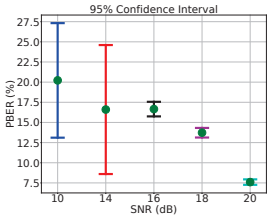
(o) VarChip at Hallway (1 m)



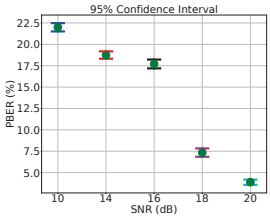
(p) VarChip at Hallway (4 m)



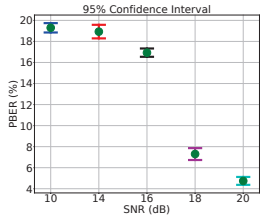
(q) VarChip at Hallway (7 m)



(r) VarChip at Hallway (10 m)

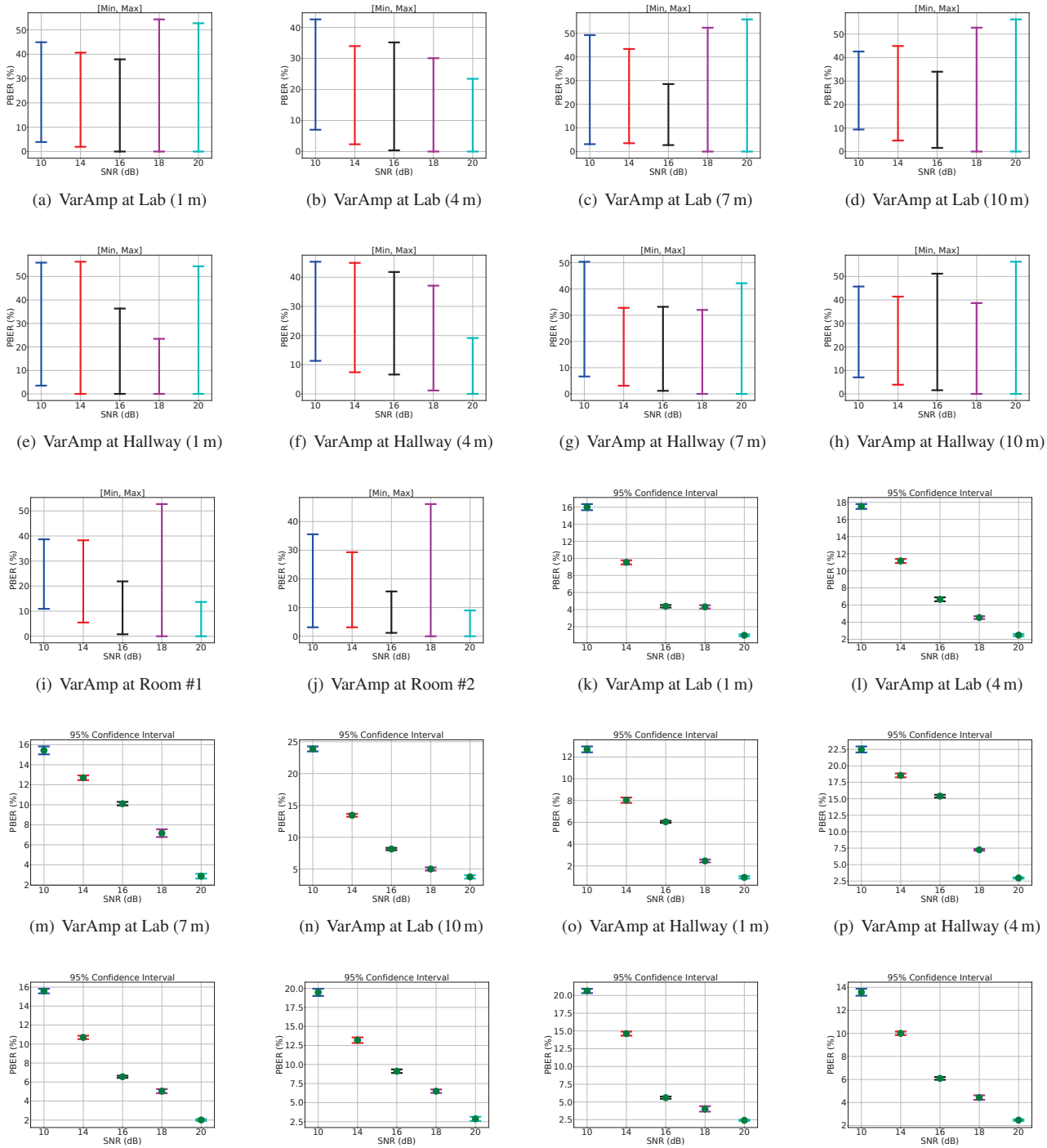


(s) VarChip at Room #1



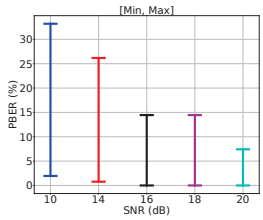
(t) VarChip at Room #2

Figure 17: Minimum, maximum and 95% confidence intervals of PBER for VarChip.

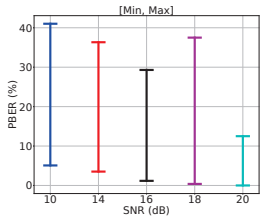


(a) VarAmp at Lab (1 m)
(b) VarAmp at Lab (4 m)
(c) VarAmp at Lab (7 m)
(d) VarAmp at Lab (10 m)
(e) VarAmp at Hallway (1 m)
(f) VarAmp at Hallway (4 m)
(g) VarAmp at Hallway (7 m)
(h) VarAmp at Hallway (10 m)
(i) VarAmp at Room #1
(j) VarAmp at Room #2
(k) VarAmp at Lab (1 m)
(l) VarAmp at Lab (4 m)
(m) VarAmp at Lab (7 m)
(n) VarAmp at Lab (10 m)
(o) VarAmp at Hallway (1 m)
(p) VarAmp at Hallway (4 m)
(q) VarAmp at Hallway (7 m)
(r) VarAmp at Hallway (10 m)
(s) VarAmp at Room #1
(t) VarAmp at Room #2

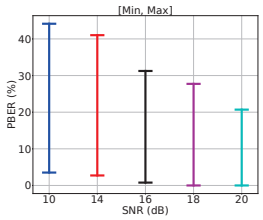
Figure 18: Minimum, maximum and 95% confidence intervals of PBER for VarAmp.



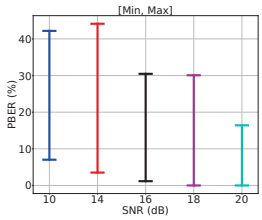
(a) VarPhase at Lab (1 m)



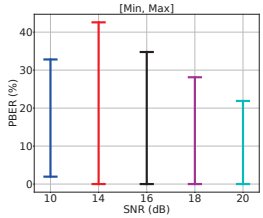
(b) VarPhase at Lab (4 m)



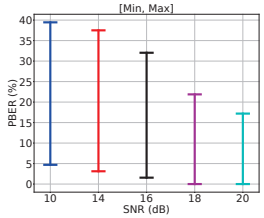
(c) Varphase at Lab (7 m)



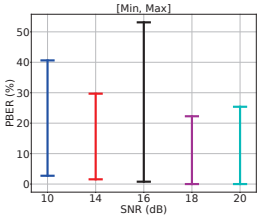
(d) Varphase at Lab (10 m)



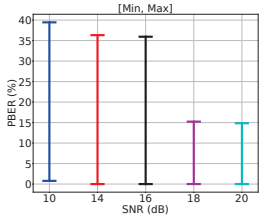
(e) Varphase at Hallway (1 m)



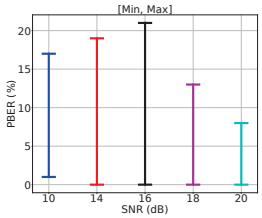
(f) Varphase at Hallway (4 m)



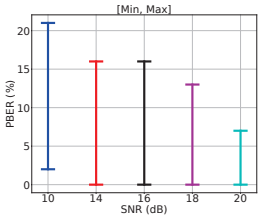
(g) Varphase at Hallway (7 m)



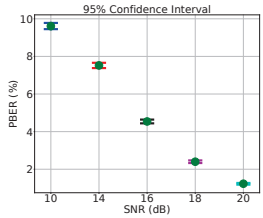
(h) Varphase at Hallway (10 m)



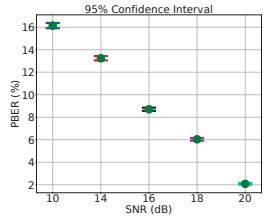
(i) Varphase at Room #1



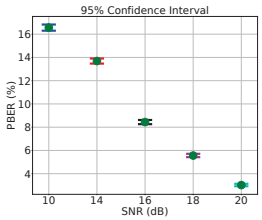
(j) Varphase at Room #2



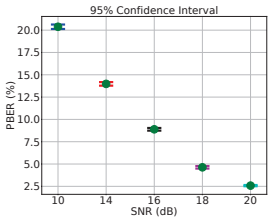
(k) Varphase at Lab (1 m)



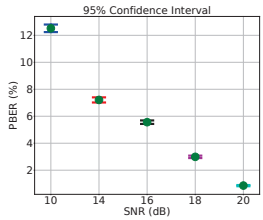
(l) Varphase at Lab (4 m)



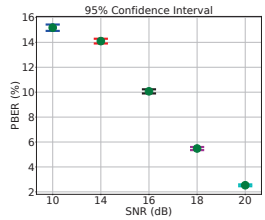
(m) Varphase at Lab (7 m)



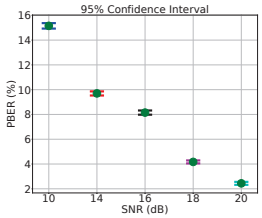
(n) Varphase at Lab (10 m)



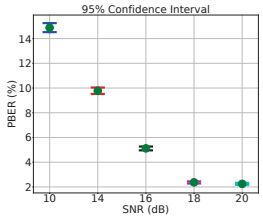
(o) Varphase at Hallway (1 m)



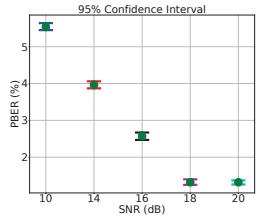
(p) Varphase at Hallway (4 m)



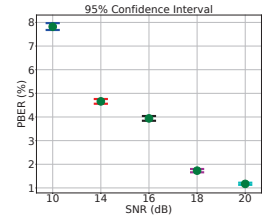
(q) Varphase at Hallway (7 m)



(r) Varphase at Hallway (10 m)



(s) Varphase at Room #1



(t) Varphase at Room #2

Figure 19: Minimum, maximum and 95% confidence intervals of PBER for VarPhase.