An Experimental Study of Vulnerabilities in the Zigbee Protocol for IoT Systems

Aidan Weingrad
Muhlenberg College
aweingrad@muhlenberg.edu

Proyash Podder
Muhlenberg College
proyashpodder@muhlenberg.edu

***Abstract*—The use of IoT devices is skyrocketing worldwide. As of 2025, there are around 20.1 billion IoT-connected devices globally—a year-over-year increase of 13% and forecasts suggest this number will rise to over 40 billion in the coming years. This meteoric growth underscores the critical importance of robust security across IoT ecosystems.**

The security of an IoT device largely depends on the communication protocol it uses, since any vulnerabilities in the protocol directly compromise the device and its data exchange. Zigbee is one of the most widely adopted communication protocols in the IoT ecosystem, yet its security has been under continuous scrutiny. Although Zigbee 3.0 introduced stronger mechanisms such as install codes to mitigate some weaknesses, many devices in the market still rely on older versions like Zigbee 1.0, which use a global trust center link key and expose significant attack surfaces.

In this work, we extend a formal security analysis of Zigbee 1.0 and 3.0 by implementing a practical packet sniffing experiment. Our findings demonstrate that even with the basic global trust center link key, it is possible to extract sensitive cryptographic material during device onboarding. These vulnerabilities have significant implications on the confidentiality and integrity of Zigbee-based systems. This paper documents our formal analysis, real-world testbed setup, and empirical findings to emphasize the continued need for rigorous security assurance in low-power wireless protocols.

***Index Terms*—Iot Security, Protocol Analysis, Vulnerability Detection, Experimental Study**

I. Introduction

The rapid proliferation of Internet of Things (IoT) devices has transformed modern technology over the last decade, driving applications from smart homes and building automation to healthcare monitoring and critical infrastructure. IoT ecosystems often rely on lightweight communication protocols that emphasize interoperability and low energy consumption, sometimes at the expense of rigorous security guarantees. Among these protocols, Zigbee [1] has emerged as one of the most widely adopted standards. Built on IEEE 802.15.4, Zigbee is designed for low-power, short-range wireless communication and supports scalable mesh networking at relatively low cost. Its widespread adoption spans domains such as smart lighting, home automation, energy management, and security systems. However, the same characteristics that drive adoption also make Zigbee an attractive target for attackers.

Prior work [2]–[6] has revealed numerous weaknesses in the Zigbee protocol. Early versions, such as Zigbee 1.0, relied on a global, pre-configured trust center link key to simplify onboarding. While this reduced deployment complexity, it created a critical vulnerability: once an adversary learned the key, they could intercept and decrypt sensitive traffic, including the network key used to protect ongoing communication. Zigbee 3.0 attempted to mitigate these risks by introducing install codes and more robust key management practices. Nevertheless, multiple studies show that weaknesses persist, and real-world implementations often introduce additional flaws due to cost constraints, limited expertise, or compatibility requirements.

In this paper, we build on prior research by bridging the gap between theoretical analysis and real-world experimentation. Using commodity hardware (TI CC2531 USB dongle, Packer Sniffer, etc.), we perform packet sniffing during the Zigbee device onboarding process. Our analysis demonstrates that even in modern implementations, sensitive cryptographic material such as the network key can still be recovered if the adversary possesses knowledge of the global trust center link key. This result underscores the continued relevance of legacy vulnerabilities, as large numbers of deployed devices remain reliant on older Zigbee versions.

Our contributions are twofold. First, we revisit formal analyses of Zigbee 1.0 and 3.0, highlighting improvements introduced in the newer standard while noting persistent gaps. Second, we design and execute a packet sniffing experiment that validates prior findings in real-world conditions, showing concretely how critical keys can be exposed during onboarding. By combining formal and empirical perspectives, our work emphasizes the importance of protocol-level scrutiny in IoT security. As ecosystems expand, attackers will increasingly target foundational communication layers to compromise large numbers of devices at once. Without strong guarantees of protocol security, higher-level safeguards such as application encryption or secure cloud integration are insufficient.

The remainder of this paper is organized as follows. Section II provides background on the Zigbee protocol, and Section III reviews related work. Section IV explains our motivation for extending prior formal analyses into the design of a packet sniffing experiment. Section V describes the experimental setup, data collection, and results. Section VI discusses the broader implications of our findings, and Section VII concludes the paper with recommendations for strengthening IoT protocol security.

II. Background

Zigbee is built upon the IEEE 802.15.4 standard, which defines the physical and MAC layers. Zigbee extends this with its own network and application layers to support low-power, low-data-rate wireless mesh networks.

Three device types exist: coordinator, router, and end device. The coordinator initiates and maintains the network, distributes keys, and manages device authorization. Routers extend the mesh, and end devices communicate through the coordinator or a router.

A. Key Management in Zigbee

Zigbee uses 128-bit AES encryption for securing messages. The key hierarchy includes:

- **Network Key:** Shared across all devices for NWK layer security.
- **Link Key:** Used for APS-layer encryption between two devices.
- **Trust Center Link Key:** Used to encrypt the network key during device onboarding.

Zigbee 1.0 typically uses a global trust center link key, `ZigbeeAlliance09`, known to all vendors. Zigbee 3.0 introduces Install Codes to derive unique per-device keys.

1) Initial Key Generation: A ZigBee network begins with the coordinator generating a network key and distributing it to joining devices. In ZigBee 1.0, this key is encrypted with a global, publicly known link key, making the process insecure. ZigBee 3.0 improves this by using *Install Codes* to derive unique per-device trust center link keys, reducing the risk of exposure [7].

2) Join a Secured Network: To join a ZigBee network, a device discovers a coordinator and requests association. The coordinator then sends the network key encrypted with the preconfigured link key. In ZigBee 3.0, this key is updated after joining to a device-specific trust center link key, strengthening security [7].

3) Application Key Establishment: For secure end-to-end communication, two devices require an application link key. The initiator requests this key from the coordinator, which generates and securely delivers it to both devices, enabling protected communication without relying on the network-wide key [7].

III. Related Work

Security of the Zigbee protocol has been widely studied, with both formal and experimental work revealing significant weaknesses. Because Zigbee builds on IEEE 802.15.4, protocol-level flaws can propagate across large IoT ecosystems that depend on it for secure communication [1], [8].

Li et al. [9] conducted one of the most comprehensive formal analyses using the Tamarin prover, showing that Zigbee 1.0 is fundamentally insecure due to its publicly known global trust center link key, which enables decryption of the network key during onboarding. Although Zigbee 3.0 introduces install codes and per-device keys, risks persist when backward compatibility with legacy devices is maintained. Other analyses have also identified vulnerabilities such as replay, orphan-frame manipulation, and denial-of-service attacks [10].

Empirical studies reinforce these findings. Shafqat et al. demonstrated with *ZLeaks* that user activities can be inferred from encrypted traffic [2], while Li et al.'s *ZPA* showed similar privacy leakage using machine learning [4]. Ren et al.'s *Z-Fuzzer* uncovered multiple implementation flaws, including CVEs, and later work introduced more advanced fuzzing techniques [3], [5]. Hardware-based analysis of Zigbee 3.0 further confirmed ongoing issues with denial-of-service and key management [6].

Together, these studies show that Zigbee's encryption and key management remain vulnerable in both theory and practice. Building on this foundation, we reproduce key formal findings and validate them experimentally through packet sniffing that demonstrates how default keys can expose critical network material.

IV. From Formal Proofs to Practical Curiosity

Our investigation into Zigbee security began with insights from prior formal work. Using the *Tamarin prover*, researchers modeled the Zigbee key establishment procedures as a set of rules that capture message flows, cryptographic operations, and trust assumptions [9]. Within this symbolic framework, the prover automatically searches for counterexamples to security properties such as confidentiality, authenticity, and key freshness. This analysis revealed that Zigbee 1.0 is vulnerable, largely because of its dependence on a global trust center link key shared across all devices. Specifically, Tamarin proved that the confidentiality of the network key is compromised during the initial key transport. Since this key is publicly known, an adversary can use it to decrypt the network key transmitted during the device onboarding process Once exposed, the network key compromises the confidentiality and integrity of the entire network, enabling eavesdropping, device impersonation, or denial-of-service attacks.

Zigbee 3.0 introduced improvements such as install codes, unique trust center link keys, and refined key transport procedures. Formal modeling of Zigbee 3.0 confirmed that these changes eliminate some of the flaws present in earlier specifications. However, our findings suggest that manufacturers may not fully embrace the newer specifications, often falling back on legacy behaviors or continuing to support devices that still implement Zigbee 1.0, thereby perpetuating known security risks. This creates a practical dilemma: although the protocol specification has evolved, the deployment landscape is still heterogeneous, with insecure versions coexisting alongside more secure ones.

While the formal proofs provide rigorous evidence of Zigbee's theoretical weaknesses, they also raised an important question for us: *Do these vulnerabilities manifest in real-world communication?* The Tamarin-based analysis demonstrates that the global trust center key undermines confidentiality in principle, but it does not capture implementation-

specific details. We were therefore motivated to complement the symbolic results with an empirical approach, focusing on observing actual Zigbee communication in a controlled testbed environment.

We designed a **packet sniffing experiment** using commodity hardware to capture Zigbee traffic during the onboarding process. By doing so, we aimed to bridge the gap between formal verification and real-world experimentation, grounding the abstract vulnerabilities in concrete evidence. The following section details our experimental setup and methodology for examining Zigbee communication in practice.

V. Practical Experiment: Sniffing and Decrypting Zigbee Traffic

In this section, we present our practical experiment designed to evaluate Zigbee's security properties in a real-world setting. Using commodity hardware and open-source tools, we set up a testbed, captured Zigbee traffic during device onboarding and operation, and analyzed the packets to assess whether encryption mechanisms can be bypassed. Our goal is to demonstrate how theoretical weaknesses translate into practical exploits under realistic conditions.

A. Hardware and Software

We used the following setup:

1) **CC2531 Dongle**: Texas Instruments USB stick with 802.15.4 radio.
2) **CC Debugger**: For flashing the dongle.
3) **ZBoss Sniffer**: A sniffer stack from DSR (DSR Corporation) [11] that interfaces with Wireshark.
4) **Raspberry pi**: We have use Raspberry pi model 4 to set up and use OpenHAB
5) **OpenHAB**: The open Home Automation Bus (openHAB) [12] is an open source, technology agnostic home automation platform which runs as the center of your smart home!
6) **Sonoff Coordinator dongle**: This acts as the cooridnator or hub to setup Zigbee network [13].
7) **Zigbee Supported Smart Devices**: We have used several smart devices (e.g., SmartPlug, Motion Sensor, etc.) for generating Zigbee traffic and perform our experiment.
8) **Wireshark**: A open-source network protocol analyzer used to capture and inspect network traffic.

B. Firmware Flashing of CC2531 Dongle

To flash the CC2531 USB dongle for our experiments, we used Texas Instruments' RF Flasher (non-V2) on a Windows machine. Firmware updates for the debugger were applied using files from the Texas Instruments website. We connected the **CC Debugger** to the **CC2531 board** using a ribbon cable and pin adapter, ensuring proper orientation by aligning the notch on the adapter with the "Debugger" label on the CC2531. The firmware for the CC2531 and the ZBoss sniffer tool were obtained from the DSR IoT tools repository. Using RF Flasher, we selected the appropriate System on Chip, loaded the *'.hex'* firmware file, and executed the Erase, Program, and Verify steps to complete the flashing process. Finally, we verified whether the flash was successful or not by confirming that the CC2531 appears under COM devices in Device Manager, then proceed to the sniffing steps.

C. Zigbee Packet Sniffing and Data Collection

To capture Zigbee communication, we set up a test environment using *OpenHAB* with a CC2531 USB dongle configured as the network coordinator. Several Zigbee-enabled devices were included in the setup, such as a Zigbee contact sensor and a SONOFF smart plug, in order to generate realistic traffic for analysis. We designed different scenarios to capture various events. For instance, we first observed the device **joining the network**, which involved the exchange of initial keys between the coordinator and the device. After this, we created traffic by simulating device-specific actions, such as **turning devices on and off**, **triggering contact sensors**, and **disconnecting and rejoining devices**. Other scenarios we considered included **broadcast messages**, **status update polling**, and **network key refresh procedures**, as these represent typical events in operational Zigbee networks.

The packet sniffing process was performed on a Windows machine with the CC2531 dongle inserted into a USB port. We employed the *ZBoss Sniffer* tool to interface with the dongle, which was successfully detected as a COM4 device. The sniffer was configured to operate on the correct frequency band and channel—specifically Page 0, Channel 11—matching the default configuration of the coordinator dongle. Upon execution, ZBoss streamed all captured packets into Wireshark in real time. For each of the defined scenarios, all communication was saved in PCAP format for detailed offline analysis, enabling us to carefully inspect packet structure, key management procedures, and potential exposure of sensitive material.

D. Analysis of Captured Zigbee Communication

We began our analysis with the packet captures obtained from the Zigbee contact sensor setup. At first glance, all packets appeared to be encrypted (see Figure 1), which aligns with Zigbee's use of AES-based encryption mechanisms.

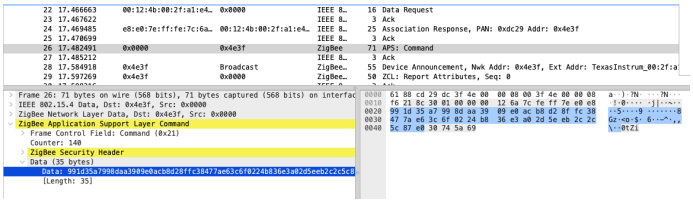

Figure 1. Wireshark Packet Capture during Contact Sensor Setup

However, as discussed in the previous section, **Zigbee 1.0 relies on the Zigbee Alliance's default global link key**, a value that is publicly documented in the specification:

```
5A 69 67 42 65 65 41 6C 6C 69 61 6E 63
65 30 39
```

Because this key is well known, an adversary can use it to attempt decryption of the initial network traffic. In our experiment, we supplied this default global key to Wireshark's automatic Zigbee decryption feature. As a result, several packets—including those exchanged during the device onboarding phase—were successfully decrypted (see Figure 2). Figure 2 just represents the decrypted version of Figure 1. This observation confirms that our device was indeed using the default global key to protect early communication, validating the theoretical weakness highlighted in formal analyses.

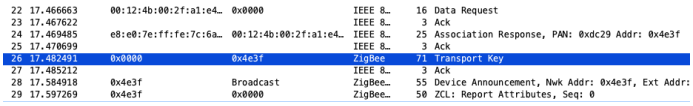

Figure 2. Wireshark Capture for the Transport Key

One of the decrypted messages, highlighted in the same capture, was labeled as a **Transport Key** frame. Upon closer inspection of the packet details in Wireshark (Figure 3), we identified a subfield named `Key`, which revealed the following 128-bit value:

`cc0b333b6c48274a220994dfbc00dae8`

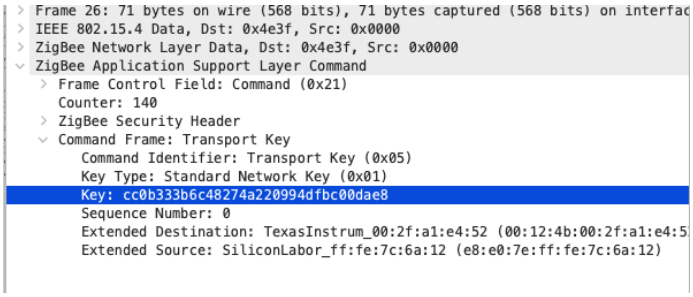

Figure 3. Detail Packet Analysis for Transport Key

This key represents the *network (transport) key* distributed by the coordinator to the device. Once established, both the coordinator and device use this transport key to encrypt all subsequent communication within the Zigbee network. In Figure 1, we have seen the decrypted version of the key, where using the *Zigbee Alliance's default global link key*, we have decrypted the communication and figured out the decrypted network key.

To further test this, we analyzed packets captured during the operation of the contact sensor, specifically events corresponding to the sensor being toggled between the *open* and *closed* states. By supplying the recovered transport key to Wireshark's decryption settings, we were able to decrypt the encrypted traffic in its entirety. As a result, we could directly interpret messages that revealed sensitive device states, such as whether a door was open or closed.

Figure 4 presents one fully decrypted ZigBee communication messages exchanged between the contact sensor and the coordinator. In the response message, the device explicitly reports its ZoneStatus attribute, which indicates that the current

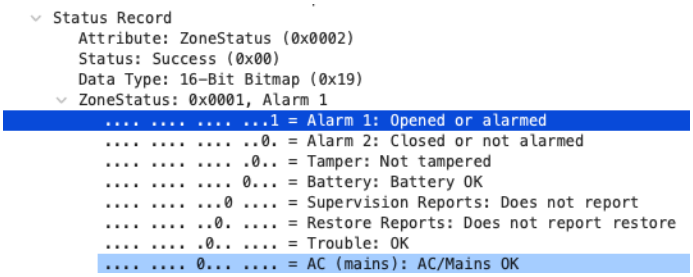

Figure 4. Decrypted Packets for a door being open

state of the door is *open*. This information is derived directly from the contact sensor's data reading, thereby confirming that the decrypted payload accurately reflects the physical state of the monitored door.

This outcome demonstrates a critical flaw: although Zigbee employs encryption, the use of a default global link key undermines its security guarantees. An adversary capable of passively capturing Zigbee traffic during the onboarding process can recover the network key and subsequently decrypt all communications. In effect, this allows attackers to monitor user behavior and access private information, thereby defeating the very purpose of encryption in securing IoT communications.

E. Implications of Key Recovery

The successful recovery of the network key in our experiments carries important implications for the security of Zigbee-based systems. At a fundamental level, this result demonstrates that the confidentiality and integrity of communications cannot be guaranteed if a device relies on the default global link key during onboarding. Once an adversary has obtained the transport key, all subsequent traffic within the network becomes vulnerable to passive decryption and active manipulation.

In practice, this means that attackers can easily infer sensitive information about users and their environments. For example, by monitoring decrypted messages, an adversary could determine when a door is opened or closed, when lights are switched on or off, or when smart appliances are operating. Such information leakage poses significant risks not only to privacy, but also to physical security—for instance, enabling adversaries to infer occupancy patterns within a home.

Moreover, the ability to intercept and use the network key opens the door to more advanced attacks. Beyond simple eavesdropping, adversaries can inject forged packets, impersonate legitimate devices, or even take control of the network by manipulating encrypted communication flows. This undermines trust in the entire IoT ecosystem and highlights the systemic weakness introduced by relying on global or default keys.

These findings underscore the necessity of stronger key management practices and strict adherence to the security mechanisms introduced in later versions of Zigbee, such as Zigbee 3.0's install code–based approach. However, given the

widespread presence of legacy devices that still implement Zigbee 1.0, the risk of key recovery attacks will persist for the foreseeable future, requiring urgent attention from both manufacturers and end users. Also due to convenience, manufacturer still uses Zigbee Version 1.0 which might increase the risks of attack surface.

VI. Discussion and Implications

This study highlights the persistent security challenges in Zigbee-based IoT systems by combining formal analyses with empirical validation. Our experiments confirmed that Zigbee 1.0, still widely deployed, remains vulnerable due to its reliance on a publicly known global trust center link key. During onboarding, this key enables an adversary to extract the network key, compromising the confidentiality and integrity of the entire network and allowing passive monitoring, behavior inference, or even device impersonation.

Although Zigbee 3.0 was designed to mitigate these issues through install codes and per-device keys, our testbed showed that real deployments may still fall back on insecure defaults. This illustrates a common IoT security problem: improvements in specifications do not guarantee secure implementation. Vendors often preserve legacy behavior for cost or compatibility reasons, leaving many devices exposed.

The implications extend beyond individual devices. Because Zigbee is deeply embedded in smart home, healthcare, and industrial systems, protocol-level weaknesses can escalate into systemic risks. Recent issues such as CVE-2024-7322 [14], involving a vulnerability in encrypted rejoin handling, demonstrate how a single flaw can trigger network-wide denial-of-service. Long device lifespans and infrequent firmware updates further exacerbate these risks.

From a broader perspective, our work underscores:

- **Legacy risk persistence** — Older Zigbee versions remain widely deployed, and backward compatibility preserves known weaknesses.
- **Protocol–deployment gap** — Security improvements in Zigbee 3.0 are inconsistently adopted.
- **Systemic threat potential** — Protocol-layer flaws can affect entire ecosystems, not just individual devices.

To address these challenges, we recommend:

- Manufacturers should eliminate global keys and enforce Zigbee 3.0's install code–based key management.
- End-users should verify device firmware and protocol versions to ensure secure onboarding.
- Researchers should extend analysis to additional Zigbee mechanisms such as commissioning and touchlink.

VII. Conclusion and Future Work

This work demonstrates that formal and empirical methods complement each other in identifying vulnerabilities in IoT protocols. By combining symbolic analysis with real-world packet sniffing, we showed that attackers can recover sensitive cryptographic material during onboarding, enabling both eavesdropping and active manipulation. Our findings highlight the need for stricter adherence to Zigbee 3.0 security mechanisms and for the rapid deprecation of Zigbee 1.0 devices.

The ongoing reliance on insecure defaults reflects a deeper systemic issue. Manufacturers prioritize cost and compatibility, users are often unaware of risks, and slow protocol migration leaves known vulnerabilities active for years. Without changes across the ecosystem, IoT deployments will remain susceptible to attacks targeting foundational communication layers.

Future work includes:

- **Protocol fuzzing and automated testing** to uncover additional flaws in Zigbee implementations.
- **Cross-layer attack studies** examining how Zigbee weaknesses interact with higher-level application vulnerabilities.
- **Deployment monitoring tools** to detect insecure onboarding or fallback to global keys.
- **Policy and standardization efforts** encouraging manufacturers to adopt secure defaults and retire outdated specifications.

References

[1] (2015) Zigbee specification. [Online]. Available: https://csa-iot.org/all-solutions/zigbee/

[2] N. Shafqat, M. Shafique, Z. Qian, Z. Shafiq, and A. Wang, "Zleaks: Passive inference attacks on zigbee-based smart homes," *arXiv preprint*, vol. arXiv:2107.10830, 2021. [Online]. Available: https://arxiv.org/abs/2107.10830

[3] J. Ren, L. Chen, Y. Liu, Z. Zhang, and J. Sun, "Security analysis of zigbee protocol implementation via device-agnostic fuzzing," *ACM Transactions on Privacy and Security (TOPS)*, vol. 26, no. 4, pp. 1–28, 2023. [Online]. Available: https://dl.acm.org/doi/full/10.1145/3551894

[4] Y. Li, M. Zhang, Y. Zhou, and J. Xu, "Zpa: A smart home privacy analysis system based on zigbee encrypted traffic," *Security and Communication Networks*, vol. 2023, pp. 1–12, 2023. [Online]. Available: https://dl.acm.org/doi/10.1155/2023/6731783

[5] J. Ren, Y. Liu, Z. Zhang, and J. Sun, "Intelligent zigbee protocol fuzzing via constraint-field dependency inference," in *International Conference on Information and Communications Security (ICICS)*. Springer, 2024, pp. 372–388. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-51476-0_23

[6] A. Ghobakhlou, N. Nair, J. Gutierrez, and S. Jamali, "A comprehensive analysis of security challenges in zigbee 3.0 networks," *Sensors*, vol. 25, no. 15, p. 4606, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/15/4606

[7] (2015) Zigbee specification. [Online]. Available: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

[8] IEEE Computer Society, "Ieee standard for low-rate wireless networks (ieee 802.15.4)," https://standards.ieee.org/standard/802_15_4-2015.html, 2015, accessed September 9, 2025.

[9] L. Li, P. Podder, and E. Hoque, "A formal security analysis of zigbee (1.0 and 3.0)," in *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*, 2020, pp. 1–11.

[10] Y. Pan, "Zigbee wireless network attack and detection," in *Advances in Computer Communication and Computational Sciences*. Springer, 2021, pp. 365–376. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-78621-2_32

[11] Dsr - doing software right. [Online]. Available: https://en.dsr-corporation.com

[12] The open home automation bus (openhab). [Online]. Available: https://www.openhab.org

[13] Sonoff zigbee 3.0 usb dongle plus. [Online]. Available: https://sonoff.tech/en-us/products/sonoff-zigbee-3-0-usb-dongle-plus-zbdongle-p

[14] National Vulnerability Database (NVD). (2025) CVE-2024-7322 Detail. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2024-7322