

PUFBind: Practical Hardware-Bound Encryption Scheme for IoT Devices against White-Box Cloning Attacks

Jun Liu^{1,2}, Yang Shi³, Jie Chen², Xiaoli Dong⁴, Lu Zhou⁵, Bo Yang^{1*}

¹School of Artificial Intelligence and Computer Science, Shaanxi Normal University, Xi'an 710119, China

²State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China

³School of Computer Science and Technology, Tongji University, Shanghai 200092, China

⁴School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, China

⁵School of Computer Science and Technology, Xidian University, Xi'an 710071, China

Keywords: Internet of things (IoT); White-box encryption; Physically unclonable function; Code cloning attack; Provable security

Citation: Liu J., Shi Y., Chen J., et al.: PUFBind: Practical Hardware-Bound Encryption Scheme for IoT Devices against White-Box Cloning Attacks. *Data Intelligence*, 2026, Vol. 8, Art. No.: 20260114. DOI: <https://doi.org/10.3724/2096-7004.di.2026.0114>

ABSTRACT

Internet of Things (IoT) devices in exposed environments face white-box attacks such as key extraction and code cloning. This paper introduces PUFBind, a cryptographic system for cloud-device architectures that binds decryption to a device's physical unclonable function. By using the physical unclonable function response as a tweak in a tweakable encryption scheme, PUFBind ensures data confidentiality for the server and resists both key extraction and cloning attacks on the device. We provide formal security proofs for device binding resistance and evaluate a practical implementation, demonstrating efficient performance on typical IoT platforms while outperforming existing white-box encryption schemes.

1. INTRODUCTION

The rapid evolution of the Internet of Things (IoT) has given rise to pervasive cloud/edge-to-device architectures, where resource-constrained endpoints like sensors, actuators, and embedded controllers communicate with powerful remote servers for management and data aggregation. In applications ranging from industrial automation and smart grids to connected healthcare, these endpoints often operate in physically exposed and unattended environments. A temperature sensor in a remote pipeline, a smart lock on a shared vehicle, or an infusion pump in a patient's home are all susceptible to physical capture and

*Corresponding author: Bo Yang (E-mail: byang@snnu.edu.cn; ORCID: 0000-0002-0419-1209).

inspection. This vulnerability fundamentally breaks the black-box assumption of traditional cryptography, where keys are presumed to be hidden within a secure boundary.

In such white-box attack contexts, adversaries can extract and analyze the device's firmware, leading to two devastating breaches, as illustrated in Figure 1. One is key extraction, where the long-term cryptographic secret is recovered, allowing decryption of all past and future communications. The other is code cloning, where the entire firmware is cloned and deployed on counterfeit devices. The latter enables large-scale device impersonation, allowing fake nodes to inject false data into the cloud or illegitimately receive sensitive commands, thereby compromising the integrity of the entire IoT system. For service providers, this translates to direct revenue loss and eroded trust.

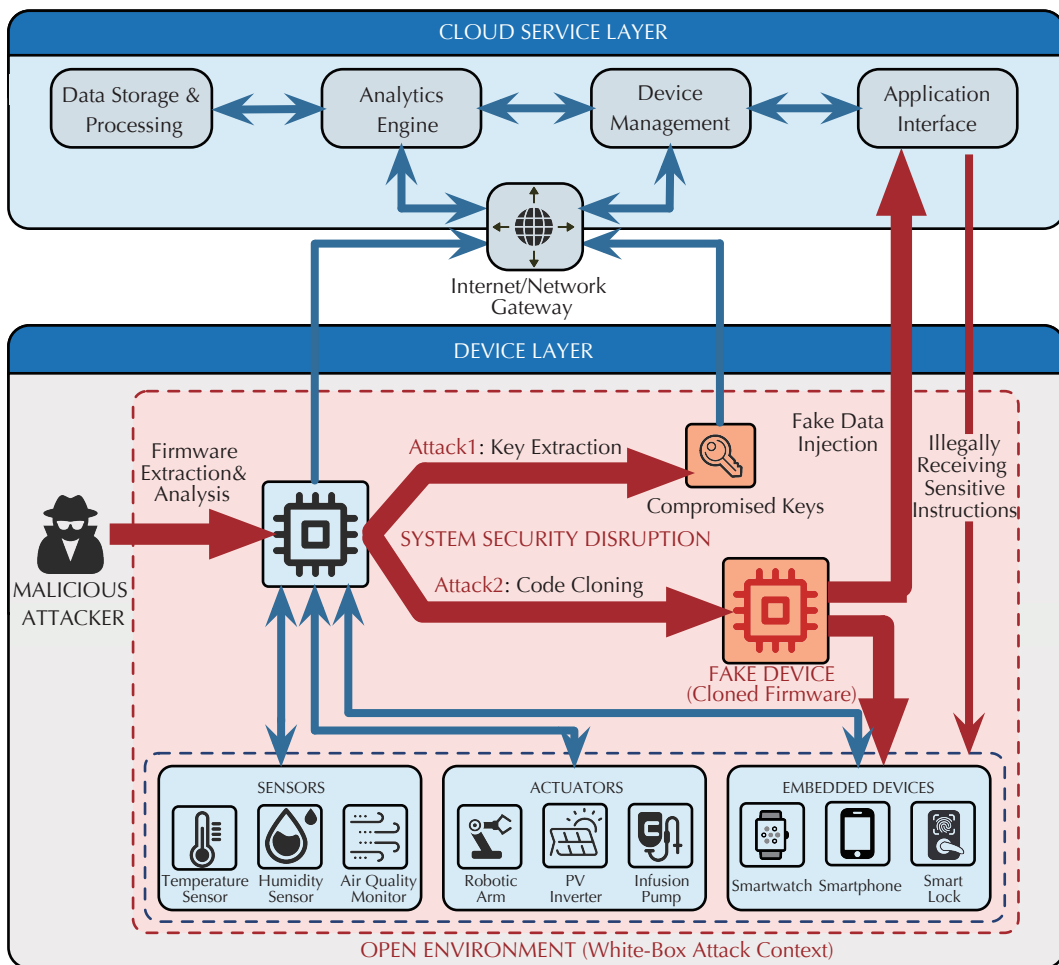


Figure 1. White-Box Attacks in IoT Cloud-Device Architecture.

Current security solutions address these threats in a fragmented manner. White-box cryptography (WBC) has been tailored for IoT endpoints to obfuscate keys within software [1]. However, these are primarily software-only solutions. While they aim to resist key extraction, they offer no protection against the physical cloning of the software itself. Conversely, hardware-based roots of trust, such as physically unclonable functions (PUFs), provide a means for device fingerprinting and binding. Pioneering works have explored PUF-assisted authentication [2] or proposed theoretical frameworks for PUF-integrated encryption [3-4]. Yet, a critical gap persists in the literature for a holistic, system-ready cryptographic scheme that is natively designed for the cloud-device paradigm, providing simultaneous protection for both the server's data confidentiality and the endpoint's resistance to cloning.

To this end, we introduce PUFBind, a cryptographic system co-designed for the IoT cloud-device architecture. PUFBind operates seamlessly within the standard communication loop, in which the server encrypts commands or data using a device-specific cryptographic parameter derived from the target device's PUF, and only that specific device can decrypt it. Our design provides a unified yet layered defense. For the cloud/server side, the system functions as a secure tweakable encryption scheme, ensuring that data encrypted for one device remains confidential to all others, meeting standard cryptographic notions of confidentiality. For the endpoint device side, the firmware is a white-box implementation that obfuscates the key. Crucially, its core decryption routine is cryptographically fused with the device's intrinsic PUF response. Possessing the binary alone is useless without the unique physical hardware. The principal contributions of this work are summarized as follows.

1. We present PUFBind, a concrete cryptographic construction for IoT cloud-device secure channels. It leverages the tweakable block cipher paradigm, where the device's PUF response acts as the tweak, creating a natural and efficient binding mechanism within established cryptographic constructs.
2. We evaluate security from perspective of each architectural component. For the server-side black-box security, we model and prove the confidentiality of the server's encryption under standard assumptions. For the endpoint white-box security, we first analyze the endpoint's resistance to key-extraction attacks based on its white-box design. Furthermore, in terms of endpoint anti-cloning security, we formally define and prove that successful firmware cloning and execution on a different device would imply breaking the fundamental unclonability of the PUF itself. This provides a rigorous, reductionist guarantee against code cloning attacks.
3. We implement PUFBind and benchmark it on platforms representing both cloud servers (x86_64 and ARM64) and typical IoT endpoints (ARM Cortex-A class). Our results demonstrate that the computational and storage overheads are acceptable for many IoT applications. Notably, PUFBind achieves an encryption throughput of 81.43 MB/s on a workstation and a decryption throughput of 35.40 MB/s on a Raspberry Pi 5.

The rest of this paper is structured as follows. Section 2 reviews the existing works. Section 3 presents the cryptographic preliminaries; Section 4 formalizes the system and security models. Section 5 details the PUFBind construction. Section 6 delivers the tripartite security analysis. Section 7 presents the performance evaluation. Section 8 concludes this paper.

2. RELATED WORK

Securing IoT devices, particularly those deployed in physically exposed environments, requires addressing dual threats, the exposure of software-embedded keys and the cloning of the software itself. Research efforts have diverged along two primary paths, one focusing on cryptographic key protection within software by white-box cryptography, and the other leveraging hardware intrinsic properties for device identification and binding. This section reviews representative works in both categories, concluding with a discussion of the remaining gap that motivates our work.

2.1 White-Box Cryptography for IoT Security

White-box cryptography aims to maintain the confidentiality of cryptographic keys even when the implementation is fully exposed to an adversary. A significant body of work has adapted white-box cryptography for resource-constrained IoT devices.

White-Box Implementation for Standardized Ciphers. Several studies have tailored white-box implementations for IoT environments, focusing on lightweight or standardized ciphers [5]. Zhou *et al.* designed implementations of KLEIN, PRESENT, and LBlock for IoT scenarios [6], while Jin *et al.* [7] and Miao *et al.* [8] both proposed white-box SM4 variants for edge agents, the latter using pseudo-random state obfuscation. Other approaches include Ma's white-box Schnorr signature for IoT key protection [9], and Bang *et al.*'s ECC implementations based on elliptic-curve precomputation [10] and residue number systems [11]. Further methods include Saha and Srinivasan's use of bijective encodings in block-cipher modes [12], Marin's non-commutative cryptography obfuscation [13], and Wang *et al.*'s dynamic white-box AES with separate table/key obfuscation [14]. These works primarily aim to obfuscate the key within a network of lookup tables and encoded operations, with security arguments often based on the complexity of reverse-engineering these encodings.

Dedicated White-Box Encryption Schemes. To move beyond implementations of existing ciphers, several dedicated white-box encryption schemes have been proposed [15]. Cho *et al.* designed two hybrid white-box constructions, F-CTR-WBC and UF-CTR-WBC, for IoT settings [16]. Shi *et al.* developed a series of lightweight schemes: an ultra-lightweight design for severely constrained devices [17], a scheme with random padding for wearable electronics [18], and a structure-based scheme to resist key extraction in distributed embedded systems [19]. Furthermore, Albricci *et al.* enhanced the key space of SPNbox, a white-box block cipher [20], by incorporating key-dependent operations [21]. Other notable designs include the SDSRS method by Shi *et al.*, which incurs a storage cost of 12 MB [1], and the Cubicle family of white-box block ciphers introduced by Shiba *et al.* for IoT applications [22].

While these schemes advance the state of white-box cryptography for efficiency and heuristic security, they operate purely in the software domain. Consequently, they offer no inherent protection against code cloning attacks. An adversary who obtains the binary can directly copy and execute it on another device, bypassing any key protection mechanism entirely. This makes them insufficient for scenarios where device uniqueness and anti-cloning are required.

2.2 Hardware-Assisted Device Security and Binding

Recognizing the limitation of software-only solutions, a second research direction seeks to bind cryptographic functionality or software execution to a specific hardware instance.

PUF-based Authentication Protocols. PUFs have been widely explored for device authentication. They provide a unique, hardware-derived fingerprint that can be used in challenge-response protocols to verify a device's identity. For example, [2] proposed an authentication scheme for IoT devices that combines PUF responses with key-hiding techniques within a white-box-like environment. Such protocols are effective for verifying "what you have" but are typically not designed as general-purpose encryption/decryption schemes for protecting ongoing data confidentiality between a server and a device.

Frameworks and Constructions for PUF-Integrated Cryptography. More closely related to our work are proposals that integrate PUFs directly into cryptographic constructions. [3] presented a generic framework for white-box encryption using PUFs, based on a tweakable Even-Mansour structure. It established a high-level security model but stopped short of providing a complete, instantiated scheme with detailed algorithms and a full implementation evaluation. Similarly, [4] explored combinations of a space-hard white-box cipher with PUFs, proposing several simple serial/parallel modes. While demonstrating a concrete combination, its security analysis remained relatively informal, and its design was coupled with a specific type of space-hard white-box cipher. Other notable works in hardware-binding include device-binding techniques for white-box cryptography in mobile payments [23-25]. These provide important security models but often lack a turnkey cryptographic construction for the IoT cloud-device paradigm.

The existing works at the intersection of PUFs and cryptography have made valuable conceptual contributions—validating the potential and proposing models. However, they often reside either at the abstract level of a framework without a ready-to-implement specification, or as an ad hoc combination with a specific cipher lacking a comprehensive security proof for the integrated system. A practically oriented, fully constructed encryption scheme with rigorous, reductionist security proofs for the binding property remains absent.

The reviewed literature reveals a clear divide. While white-box cryptography provides tools for key obfuscation, it fails to address physical software cloning. Conversely, hardware-assisted security offers concepts for device binding but lacks complete, proven encryption constructions suitable for the standard IoT cloud-device model. Consequently, a significant gap persists. No existing scheme provides a full server-client specification, simultaneously resists key extraction through white-box techniques, delivers provable security against code cloning via hardware-binding with security reduced to standard PUF properties, and is accompanied by a comprehensive implementation and performance analysis for real-world deployment. Our proposed scheme PUFBind is designed to fill this gap. It moves beyond a generic framework to provide a concrete construction, leverages established tweakable block cipher structures for clean integration, and centers its core security contribution on a rigorous, reductionist proof of device-binding security.

3. PRELIMINARIES

This section introduces PUFs and tweakable block ciphers, the two fundamental cryptographic primitives that underpin the design and security analysis of PUFBind. Notations used in this paper are listed in Table 1.

Table 1. Notations.

Symbol	Description
Φ	PUF system (PUF combined with fuzzy extractor)
x, z, z', h	PUF challenge, PUF response, reconstructed PUF response, PUF helper data
$\perp$	Empty initial helper data (setup mode)
$\mathcal{K}, \mathcal{T}, \mathcal{M}$	Key space, tweak space, message space
$E, \tilde{E}$	Conventional block cipher, tweakable block cipher
$\mathcal{H}; h_1, h_2$	ϵ -AXU ₂ hash function family; hash functions
Π_{WB}	The proposed white-box block cipher
$m, c, \mathcal{A}, \lambda$	plaintext, ciphertext, adversary, security parameter
k, k_1, k_2, k_3, k_4	Master key
K, K_1, K_2, K_3, K_4	White-box equivalent key
T_i	i -th 8-bit random permutation
$\tau_i, \tau'_i; L$	Non-linear transformation at round i ; linear transformation

3.1 Physically Unclonable Functions

A *physically unclonable function (PUF)* is a physical device that exploits intrinsic manufacturing variations to produce a unique, unpredictable response y for a given challenge x . Due to inherent noise, a *fuzzy extractor* [26] is used to reliably reproduce a stable cryptographic output z from noisy PUF measurements. We model the combined PUF and its associated fuzzy extractor as a *PUF system* Φ . Φ operates in two modes: the *setup mode* and the *reconstruction mode*. The setup mode is invoked the first time a challenge is used with an empty initial helper data denoted by $\perp$, and produces an output z and helper data h : $(z, h) \leftarrow \Phi(x, \perp)$. The reconstruction mode is invoked subsequently with the same challenge and the stored helper data h , regenerates the same output z (note that the helper data h is output unchanged for consistency of notation): $(z, h) \leftarrow \Phi(x, h)$.

The PUF characteristic most relevant to the scheme in this paper is selective unclonability [27], which is defined as follows. This property forms the foundation for our later reduction proof of device-binding security.

Definition 1 (Selective Unclonability). A PUF system Φ is (q, ϵ) -selectively unclonable if any adversary making at most q queries to Φ cannot produce another PUF system Φ' such that with probability greater than ϵ , Φ' is a clone of Φ ; i.e., for every challenge x ,

$$\Pr[z' = z \mid (z', h) \leftarrow \Phi'(x, h), (z, h) \leftarrow \Phi(x, \perp)] \leq \epsilon \quad (1)$$

3.2 Tweakable Block Ciphers and the CLRW2 Construction

A tweakable block cipher $\tilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ is a family of permutations indexed not only by a key $K \in \mathcal{K}$ but also by an auxiliary public input called a *tweak* $T \in \mathcal{T}$. Its security is defined as that of a *strong tweakable pseudorandom permutation (STPRP)* with respect to both the key and the tweak.

A widely-adopted and provably secure tweakable block cipher construction is CLRW2 [28], as shown in Figure 2. Let $E : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$ be a conventional block cipher and let h_1, h_2 be two hash functions drawn from a ϵ -almost-2-XOR-universal (ϵ -AXU₂) hash function family $\mathcal{H}$. The CLRW2 construction is defined as

$$\text{CLRW2}^{[E, h_1, h_2]}(T, M) = E_{K_2}(E_{K_1}(M \oplus h_1(T)) \oplus h_1(T) \oplus h_2(T)) \oplus h_2(T) \quad (2)$$

Landecker et al. proved that if E is an SPRP and H is an 2^{-n} -AXU₂ hash family, then CLRW2 is an STPRP [28].

Definition 2 (ϵ -AXU₂ Hash Family). A family of hash functions $\mathcal{H} = \{h : \mathcal{T} \rightarrow \mathcal{M}\}$ is ϵ -AXU₂ if for all distinct $T, T' \in \mathcal{T}$ and any $\Delta \in \mathcal{M}$,

$$\Pr_{h \leftarrow \mathcal{H}} [h(T) \oplus h(T') = \Delta] \leq \epsilon \quad (3)$$

The CLRW2 paradigm provides a modular and analyzable way to build a tweakable cipher. In Section 5 we will instantiate this paradigm with a white-box block cipher and a PUF-derived tweak, thereby obtaining the core of PUFBind.

4. SYSTEM MODEL AND SECURITY MODEL

This section formalizes the IoT architecture we consider, the capabilities of the adversary, and the precise security goals that our scheme aims to achieve.

4.1 System Model

We consider a typical IoT cloud-device architecture consisting of two entities: a trusted cloud server $\mathcal{S}$ and a set of resource-constrained IoT devices $\{\mathcal{D}_i\}$. Each device $\mathcal{D}_i$ is equipped with a unique PUF system Φ_i as defined in Section 3.1. The server maintains a confidential database that stores, for each registered device, a set of challenge-output pairs (x, z) generated during an initial, secure enrollment phase.

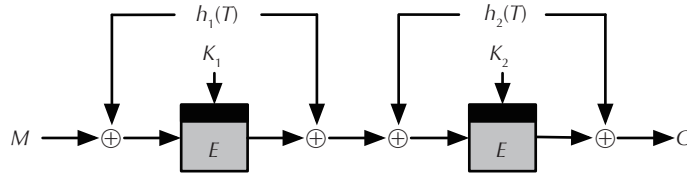


Figure 2. The CLRW2 Construction.

As illustrated in Figure 3, the operational workflow consists of three phases:

1. *Setup & Enrollment*. The server interacts with each device in a secure environment to record a number of PUF challenge-output pairs (x, z) . The server stores these pairs securely. The device retains no persistent secret other than its physical hardware.
2. *Encryption (Server-side)*. When the server wishes to send a confidential message m to a particular device $\mathcal{D}_i$, it selects a fresh, unused challenge x from its database, retrieves the corresponding PUF output z , and encrypts m using a tweakable encryption algorithm where z serves as the tweak. The resulting ciphertext c is transmitted to the device with x .
3. *Decryption (Device-side)*. Upon receiving (c, x) , the device queries its local PUF system Φ_i with challenge x to reconstruct the output z' . It then uses z' as the tweak to decrypt c and recover the message m .

This model reflects a wide range of IoT applications, such as remote control of industrial actuators, secure firmware updates, or confidential data delivery to field sensors.

4.2 Threat Model

We consider a powerful, physically-present adversary $\mathcal{A}$ whose capabilities and limitations are defined as follows.

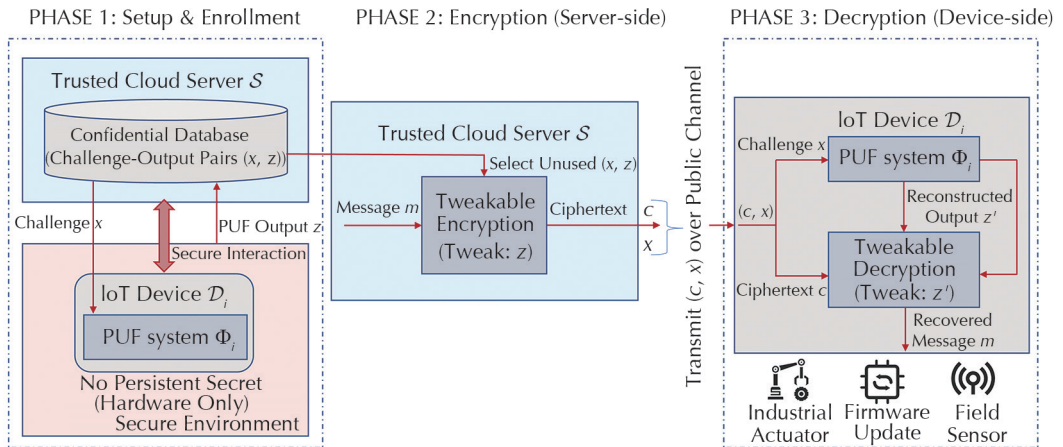


Figure 3. System Model.

First, the adversarial capabilities are enumerated below.

1. $\mathcal{A}$ can physically capture one or more IoT devices and obtain full read/write access to their memory, which is referred to as white-box access. This allows $\mathcal{A}$ to statically analyze the device's firmware, including any embedded lookup tables, encoded operations, and program logic; or dynamically observe execution states, intermediate values, and memory contents during runtime.
2. $\mathcal{A}$ can eavesdrop on all communication between the server and any device.
3. $\mathcal{A}$ can clone (copy) the entire software image of a captured device and attempt to execute it on another hardware platform.

Second, the adversarial limitations are listed below, which constitute the security assumptions of our scheme.

1. $\mathcal{A}$ cannot compromise the trusted server or its confidential database. The server's software, keys, and stored PUF data are out of $\mathcal{A}$'s reach.
2. $\mathcal{A}$ cannot physically clone the PUF of a legitimate device. The PUF's inherent selective unclonability (Definition 1) is assumed to hold.
3. $\mathcal{A}$ has bounded computational resources modeled by a system security parameter λ , e.g., $\lambda = 128$.

Finally, the specific attack steps and objectives of the adversary are outlined as follows. The adversary's actions naturally fall into two sequential stages:

1. *Stage 1: Key Extraction Attack.* After capturing a device, $\mathcal{A}$'s first objective is to extract the long-term cryptographic secret, i.e., the white-box protected key, from the device's memory.
2. *Stage 2: Code Cloning Attack.* If Stage 1 succeeds, or even if it only obtains the complete binary, $\mathcal{A}$'s next objective is to copy the software and run it on a different physical device lacking the original PUF to illegitimately decrypt future server-to-device communications or impersonate a legitimate device.

4.3 Security Notions

Corresponding to the threat model, we define three complementary security goals.

4.3.1 Server-Side Confidentiality

From the server's perspective, the encryption scheme should function as a secure tweakable block cipher. We require the construction to be an STPRP.

Let $\Pi = (\text{KGen}, \text{PEnc}, \text{PDec})$ be a tweakable block cipher. For any probabilistic polynomial-time adversary $\mathcal{A}$, the STPRP advantage is defined as:

$$\text{Adv}_{\mathcal{A}, \Pi}^{\text{STPRP}}(\lambda) = \left| \Pr \left[\mathcal{A}^{\text{PEnc}_K(\cdot, \cdot), \text{PDec}_K(\cdot, \cdot)} = 1 \right] - \Pr \left[\mathcal{A}^{\tilde{P}(\cdot, \cdot), \tilde{P}^{-1}(\cdot, \cdot)} = 1 \right] \right| \quad (4)$$

where $K \leftarrow \text{KGen}(1^\lambda)$ and $\tilde{P}$ is a uniform random tweakable permutation over the same domain. Here, λ is the security parameter, and 1^λ denotes a bit string of λ ones. We say Π is STPRP-secure if for all such $\mathcal{A}$, $\text{Adv}_{\mathcal{A}, \Pi}^{\text{STPRP}}(\lambda)$ is negligible in λ .

This notion implies both indistinguishability under chosen-plaintext attack (IND-CPA) and indistinguishability under chosen-ciphertext attack (IND-CCA) security when the tweakable block cipher is used for encryption.

4.3.2 Client-Side Key-Extraction Resistance

We require that the white-box implementation of the cryptographic algorithm on the device does not expose the underlying secret key even under the white-box access described in Section 4.2. Because a fully formal proof of white-box key extraction security is currently infeasible for practical constructions, we adopt the practical security assessment approach. We require that the design follow established white-box countermeasures, e.g., complex encodings, large state, non-linear layers, and show resilience against known differential, linear, algebraic, and dynamic attacks, etc..

4.3.3 Client-Side Device-Binding Security

This is the novel security goal that our scheme is designed to achieve. We define it via an experiment that captures the scenario where an adversary, having obtained the white-box software, attempts to make it functional on a device lacking the original PUF.

Definition 3 (Non-Functionality under Cloning Attack (NF-CA)). *Let Π be a PUF-based encryption scheme. The experiment $\text{Exp}_{\mathcal{A}, \Pi}^{\text{NF-CA}}(\lambda)$ models the scenario where an adversary attempts to make cloned software functional on a device with a different PUF:*

1. *The challenger generates system parameters and provides the adversary $\mathcal{A}$ with the white-box implementation of the decryption algorithm (excluding the PUF hardware).*
2. *$\mathcal{A}$ adaptively queries the PUF system of a target device, making at most q queries in total.*
3. *$\mathcal{A}$ outputs a complete software clone intended to run on a different hardware device.*
4. *The challenger selects a fresh challenge x^* which is never queried by $\mathcal{A}$, encrypts a uniformly random message m using the original PUF's output $z^* = \Phi(x^*, \perp)$, and sends the ciphertext c to $\mathcal{A}$.*
5. *$\mathcal{A}$ executes the cloned software on a device equipped with a different, independent PUF $\Phi' \neq \Phi$ and attempts to decrypt c to obtain $\tilde{m}$.*

The adversary $\mathcal{A}$ wins the experiment if $\tilde{m} = m$. We define the advantage of $\mathcal{A}$ against Π in the NF-CA experiment as:

$$\text{Adv}_{\mathcal{A}, \Pi}^{\text{NF-CA}}(\lambda) = \Pr[\text{Exp}_{\mathcal{A}, \Pi}^{\text{NF-CA}}(\lambda) = 1] \quad (5)$$

The scheme Π is said to be (q, ϵ) -NF-CA secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ making at most q PUF queries, $\text{Adv}_{\mathcal{A}, \Pi}^{\text{NF-CA}}(\lambda) \leq \epsilon(\lambda)$, where $\epsilon(\lambda)$ is a negligible function in the security parameter λ .

Intuitively, NF-CA models the scenario where the adversary has cloned the software but lacks the original PUF. In Section 6 we will prove that the security of this notion can be reduced directly to the selective unclonability of the underlying PUF (Definition 1).

5. THE PUFBIND CONSTRUCTION

This section presents the complete specification of PUFBind. We first explain the design rationale, then describe the white-box block cipher used as its core component, and finally give the formal algorithms along with an illustrative execution flow.

5.1 Design Rationale

The core idea of PUFBind is to cryptographically bind the decryption functionality to a specific hardware device by leveraging the tweakable block cipher paradigm. Recall from Section 3.2 that a tweakable block cipher accepts an additional public input called a tweak. We naturally let the device-specific PUF response serve as this tweak. Consequently, the encryption/decryption operation becomes intrinsically tied to the physical hardware. The same plaintext encrypted for two different devices with different PUF responses will produce distinct ciphertexts, and decryption will succeed only if the tweak used matches the one employed during encryption.

We instantiate this idea with the CLRW2 construction presented in Equation (2), a provably secure tweakable block cipher. In our setting, the underlying block cipher E is replaced by a white-box block cipher Π_{WB} which is designed to resist key extraction, and the two hash functions h_1, h_2 are instantiated using Π_{WB} itself to maintain code simplicity and a single cryptographic core. The PUF-derived tweak is integrated through these hash functions, ensuring that any change in the PUF response results in a completely different encryption path.

Tweakable Block Cipher as the Architectural Foundation. The adoption of a tweakable block cipher is fundamental to PUFBind's security model. Unlike traditional white-box schemes that rely solely on software obfuscation, our approach establishes a *hardware-software co-design* paradigm. The tweak input of the tweakable block cipher provides a natural, cryptographically sound interface for incorporating the PUF response into each encryption operation. This enables a clean security reduction to the security of the underlying tweakable block cipher, unlike ad-hoc methods of binding hardware signals to cryptographic computations, which often lack rigorous analysis. The CLRW2 construction was selected because it is a provably secure tweakable block cipher with beyond-birthday-bound security, which ensures that PUFBind remains secure even after a large number of encryptions under adversarially chosen tweaks.

5.2 Building Block: The White-Box Block Cipher Π_{WB}

Before presenting the full scheme, we describe the white-box block cipher that forms its cryptographic core. Π_{WB} is a 128-bit block cipher with a 32-round substitution-permutation network (SPN) structure, specifically designed for efficient table-based implementation and resistance against key-extraction attacks.

- Key generation algorithm $KGen() \rightarrow k$: A 128-bit master key k is chosen uniformly at random.
- Key compilation algorithm $KCom(k) \rightarrow K$: The master key k is expanded into an *equivalent key* K consisting of 128 independent 8-bit random permutations (S-boxes). This is achieved by:
 1. Using k as a seed for the standardized CTR_DRBG [29] deterministic random bit generator to produce a long pseudo-random stream R .
 2. Applying the Fisher-Yates-Knuth-Yao shuffling algorithm [30] to transform segments of R into 128 uniformly random permutations $T_1, T_2, \dots, T_{128}$ over $\mathbb{F}_{2^8}$.
 3. The set $K = \{T_1, \dots, T_{128}\}$ constitutes the white-box equivalent key.
- Encryption algorithm $Enc(K, m) \rightarrow c$: For a 128-bit plaintext m , viewed as four 32-bit words (m_0, m_1, m_2, m_3) , the encryption process executes 32 rounds of the following transformation for $i = 0, 1, \dots, 31$:

$$m_{i+4} = m_i \oplus \tau_i \circ L \circ \tau_i(m_{i+1} \oplus m_{i+2} \oplus m_{i+3}) \quad (6)$$

where:

- τ_i is a non-linear transformation that applies four 8-bit S-boxes from K :

$$\tau_i(a_0, a_1, a_2, a_3) = (T_{4i+1}(a_0), T_{4i+2}(a_1), T_{4i+3}(a_2), T_{4i+4}(a_3)) \quad (7)$$

- L is a linear diffusion layer defined for a 32-bit input b as:

$$L(b) = b \oplus (b \lll 2) \oplus (b \lll 10) \oplus (b \lll 18) \oplus (b \lll 24) \quad (8)$$

where $\lll$ denotes left circular shift.

After 32 rounds, the ciphertext is output as $c = (m_{35}, m_{34}, m_{33}, m_{32})$.

- Decryption algorithm $Dec(K, c) \rightarrow m$: Decryption is the inverse of the encryption process. For a ciphertext $c = (c_0, c_1, c_2, c_3)$, the state is updated for $i = 0, 1, \dots, 31$ as:

$$c_{i+4} = c_i \oplus \tau'_i \circ L \circ \tau'_i(c_{i+1} \oplus c_{i+2} \oplus c_{i+3}) \quad (9)$$

where τ'_i is a non-linear transformation similar to τ_i but with different order of S-boxes:

$$\tau'_i(a_0, a_1, a_2, a_3) = (T_{4(31-i)+1}(a_0), T_{4(31-i)+2}(a_1), T_{4(31-i)+3}(a_2), T_{4(31-i)+4}(a_3)) \quad (10)$$

After 32 rounds, the plaintext is recovered as $m = (c_{35}, c_{34}, c_{33}, c_{32})$.

The design of Π_{WB} emphasizes a large state, heavy use of random non-linear encodings, and deep dependency between the master key and the generated lookup tables. These features collectively increase the complexity of key recovery via algebraic analysis, BGE-style structural attacks, and dynamic code-analysis techniques commonly considered in white-box cryptography. While we do not claim a formal proof of white-box security for Π_{WB} alone, its construction adheres to established design principles aimed at practical resistance against known attack vectors in the white-box model.

5.3 Detailed Construction of PUFBind

PUFBind concretely implements the system model described in Section 4.1. It instantiates the tweakable encryption algorithm using the CLRW2 construction built upon the white-box block cipher Π_{WB} . The scheme is fully specified by the following four algorithms, which are executed within the three phases of the system model. Its overall execution flow is illustrated in Figure 4.

- Key generation algorithm $\text{PKGen}() \rightarrow k$: Executed by the server $\mathcal{S}$ during the *Setup & Enrollment* phase. The server runs $\Pi_{WB}.\text{KGen}()$ four times to obtain four independent 128-bit master keys k_1, k_2, k_3, k_4 . It outputs the combined key $k = (k_1, k_2, k_3, k_4)$ and stores it securely in its database, associated with the enrolled device $\mathcal{D}_i$.
- Key compilation algorithm $\text{PKCom}(k) \rightarrow K$: Also executed by $\mathcal{S}$ during the *Setup & Enrollment* phase, immediately after PKGen . For each master key k_i ($i = 1, \dots, 4$), the server runs $\Pi_{WB}.\text{KCom}(k_i)$ to obtain the corresponding white-box equivalent key K_i , i.e., the set of randomized lookup tables. The overall compiled white-box key is $K = (K_1, K_2, K_3, K_4)$. The server then securely transfers and embeds K into the device $\mathcal{D}_i$'s firmware or a protected storage area. After this step, the device holds the persistent white-box decryption code, and the server retains the master keys k alongside the PUF challenge-output pairs (x, z) .
- Encryption algorithm $\text{PEnc}(K, x, m) \rightarrow c$: Executed by $\mathcal{S}$ during the *Encryption* phase. When the server wishes to send a confidential message $m \in \{0, 1\}^{128}$ to device $\mathcal{D}_i$, it executes the following procedures:

1. Select a fresh, unused challenge x from its database, retrieves the corresponding PUF output z where $z = \Phi_i(x, \perp)$ and the master keys k (or a pre-computed version of K) for that device.
2. Compute the tweak-dependent values, acting as the hash functions in CLRW2:

$$\Delta_1 = \Pi_{WB}.\text{Enc}_{K_3}(z), \quad \Delta_2 = \Pi_{WB}.\text{Enc}_{K_4}(z) \quad (11)$$

3. Apply the CLRW2 structure (cf. Eq. (2)):

$$c = \Pi_{WB}.\text{Enc}_{K_2}(\Pi_{WB}.\text{Enc}_{K_1}(m \oplus \Delta_1) \oplus \Delta_1 \oplus \Delta_2) \oplus \Delta_2 \quad (12)$$

4. Output the ciphertext c , and the challenge x is sent alongside c to the device $\mathcal{D}_i$.

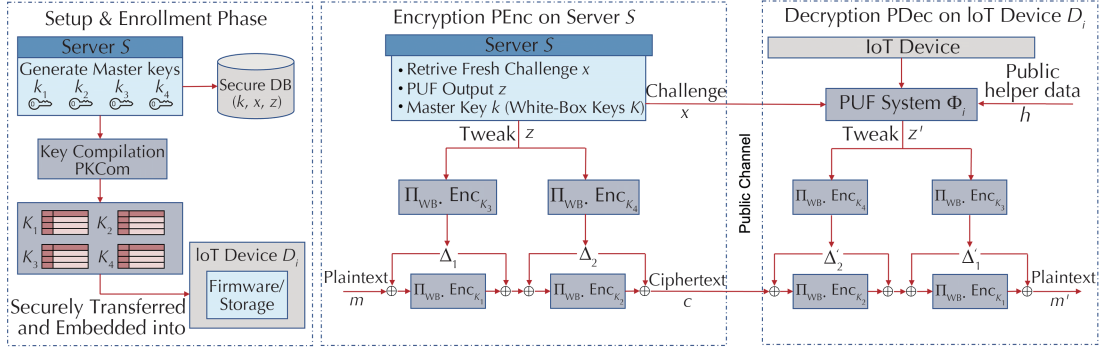


Figure 4. Execution Flow of PUFBind.

- Decryption algorithm $\text{PDec}(K, x, c) \rightarrow m$: Executed by device $\mathcal{D}_i$ during the *Decryption* phase. Upon receiving (c, x) , the device uses its embedded white-box key K and local PUF Φ_i as follows:
 - Locally query the device's PUF system with challenge x and the corresponding public helper data h to reconstruct $z' = \Phi(x, h)$.
 - Compute the tweak-dependent values using the embedded tables:

$$\Delta_{1'} = \Pi_{\text{WB}} \cdot \text{Enc}_{K_3}(z'), \quad \Delta_{2'} = \Pi_{\text{WB}} \cdot \text{Enc}_{K_4}(z') \quad (13)$$

- Invert the CLRW2 structure:

$$m' = \Pi_{\text{WB}} \cdot \text{Dec}_{K_1}(\Pi_{\text{WB}} \cdot \text{Dec}_{K_2}(c \oplus \Delta_{2'}) \oplus \Delta_{2'} \oplus \Delta_{1'}) \oplus \Delta_{1'} \quad (14)$$

- Output the recovered plaintext m' .

Execution Flow. Figure 4 depicts the complete process. The server (left) retrieves the PUF response z for the target device, encrypts the plaintext m using the CLRW2-based algorithm with z as the tweak, and transmits (c, x) . The IoT device (right) reconstructs z' from its local PUF using public helper data, then runs the white-box decryption routine, a fixed sequence of table lookups and XORs that is hard-coded to depend on z' . Crucially, the PUF system Φ is the only component that remains outside the white-box code and is executed in hardware. If $z' = z$, decryption yields m ; otherwise, the output is indistinguishable from random, thereby achieving the device-binding security defined in Section 4.3.3.

5.4 Practical Deployment Considerations

While Section 5.3 provides a complete cryptographic specification, practical deployment introduces several optimizations and design choices that enhance efficiency without compromising security.

One-Time PUF Enrollment. The enrollment procedure invokes setup mode once per provisioned challenge, and the encryption algorithm of PUFBind only retrieves the stored response for each encryption. For each challenge x in a predefined set, the PUF generates output z and public helper data h . The server stores the sensitive pair (x, z) in its secure database, forming the device's PUF profile. Subsequent encryption operations require only a fast table lookup to retrieve z , shifting computational burden to the resource-rich server, and this is an important optimization for energy-constrained IoT devices.

Challenge Management for Forward Secrecy. PUFBind's requirement for a fresh (x, z) pair per encryption provides a valuable security benefit: built-in forward secrecy for data confidentiality. The server achieves this by selecting an unused pair from the PUF profile for each encryption. The pair is kept temporarily and only discarded after the device confirms successful decryption. If decryption fails and a retransmission is needed, the same pair is reused for the retry. Only upon confirmation is the pair permanently discarded. If retries exceed a limit without success, the pair is invalidated and replaced. This ensures that even if a device's current state is compromised, previously transmitted ciphertexts remain secure, which is a property difficult to achieve with static keys in resource-constrained environments.

Challenge-Response Pair Exhaustion and Re-enrollment. PUFBind consumes one challenge-response pair (CRP) per encrypted message to guarantee forward secrecy. With a typical budget of 10^5 CRPs per device, the scheme is well-suited for low-to-moderate frequency IoT applications, as quantified in Table 2. When a trusted device approaches CRP exhaustion, re-enrollment during routine maintenance can replenish the CRP pool, allowing continued operation without compromising forward secrecy. For high-frequency communication scenarios where the CRP budget may become a constraint, several extensions could be explored to further enhance the scheme's applicability. These include using multiple pre-provisioned CRP pools, deriving key chains from each CRP to support more encryptions per pair, or combining PUFBind with session key establishment to reduce CRP consumption while preserving device binding. We leave the detailed design and analysis of such extensions for high-frequency use cases as future work.

Security-Conscious PUF Selection. Recent cryptanalytic advances have revealed vulnerabilities in many *strong* PUF constructions that support exponential challenge spaces. In light of this, PUFBind is designed to work optimally with *secure weak* PUFs, e.g., SRAM PUFs, Butterfly PUFs, which offer a limited but substantial number of independent challenges (typically 10^3 – 10^5). This design prioritizes analytical security guarantees over unbounded challenge space. The finite challenge set establishes a natural security budget for each device, which suffices for most IoT applications like firmware updates and critical command transmission.

For scenarios requiring extreme longevity, the enrollment protocol can be repeated during maintenance intervals, or PUFBind can serve as a secure bootstrap mechanism for conventional cryptographic protocols. When the device is trusted (e.g., during routine maintenance), re-enrollment can be performed securely over the existing encrypted channel: the server sends a fresh set of challenges encrypted under the current session key, the device generates new responses and returns them over the same channel, and the

Table 2. Encryption Demands of IoT Scenarios (10-year Lifetime).

Scenario	Operation Frequency	Total Encryptions	Redundancy
Smart meter	once per hour	87,600	12.4%
Insulin pump	16 times/day	58,400	41.6%
Asset-tracking tag	every 4 hours	21,900	78.1%
Smart agriculture	once per hour	87,600	12.4%

server updates the device’s CRP profile. However, if device compromise is suspected (e.g., after physical capture), re-enrollment cannot rely on the existing channel, as an adversary may have extracted the device’s keys. In this case, re-enrollment must occur in a physically trusted environment, such as a factory reset or an authorized maintenance facility. A fresh master key and a new set of challenge–response pairs are generated, the white-box lookup tables are recompiled and securely flashed onto the device, and the server revokes all previously stored credentials associated with this device. This ensures that even if the device was compromised, the adversary cannot obtain the new credentials, and the PUF’s selective unclonability prevents cloning of the device itself.

Handling PUF Noise and Decryption Failures. Our security proof assumes an ideal PUF model for simplicity. In practice, however, PUF responses are subject to noise due to environmental variations such as temperature fluctuations, voltage changes, and device aging. To address this, the fuzzy extractor [26] is employed during decryption: given a challenge x and the public helper data h , it reconstructs $z' = \Phi(x, h)$. When the reconstruction succeeds, i.e., $z' = z$, decryption proceeds normally and outputs the correct plaintext. When reconstruction fails, i.e., $z' \neq z$, the decryption output is incorrect. The fuzzy extractor guarantees that reconstruction succeeds with probability $1 - \delta$, where δ is the bit error rate (e.g., $\delta < 0.8\%$ for the SRAM PUF configuration in [31]).

For a device performing N encryptions over its lifetime, the expected number of decryption failures is approximately $N \cdot \delta$. Taking the configuration from Table 2, a smart meter performing 87,600 encryptions over 10 years would experience at most $87,600 \times 0.008 = 700$ failures, or roughly 70 failures per year. An insulin pump with 58,400 encryptions would experience at most 467 failures over its lifetime, or fewer than one per week. These failure rates are sufficiently low that manual intervention is not required.

When a decryption failure occurs, the device must detect it. This can be achieved by having the server include a message authentication code (MAC), such as HMAC over the ciphertext, which the device verifies after decryption. Alternatively, if the plaintext has a known structure (e.g., protocol headers), the device can validate these fields to detect errors without additional cryptographic overhead.

Upon detecting a failure, the device requests retransmission with the same challenge x . Since most PUF noise is transient, the second reconstruction attempt succeeds with probability $1 - \delta$, independent of the first attempt. The probability that two consecutive attempts both fail is δ^2 , which for $\delta = 0.008$ is 6.4×10^{-5} , meaning that among the expected 700 failures, only about 0.04 cases would require a third retry. In practice, a small fixed retry limit (e.g., 3 attempts) suffices to resolve virtually all transient errors. If failures persist beyond the retry limit for the same challenge x , the server marks this CRP as unreliable and falls back to a fresh pair from the device's CRP pool. Persistent failures may indicate device aging or a permanent environmental change; in such cases, the device can initiate the re-enrollment to obtain a fresh set of CRPs.

Side-Channel Attack Considerations. The PUF query $\Phi(x, h)$ during decryption processes a fresh, publicly known challenge x for each encryption. Consequently, the power consumption trace differs for each decryption operation due to the varying challenge x . Classical differential power analysis (DPA) relies on accumulating many traces under a fixed unknown secret to amplify statistical correlations. The inherent variability of the PUF input prevents this attack from converging. Similarly, differential fault analysis (DFA) requires inducing faults in a stable computation path, but the one-time nature of each PUF response leaves no persistent secret to extract. Therefore, standard side-channel attacks such as DPA and DFA are not applicable to the PUF query in PUFBind.

Separation of Concerns. This work focuses on the cryptographic construction and its formal security properties. While the secure management of the server's PUF profile database is essential for overall system security, it represents a distinct engineering challenge addressed by standard enterprise security practices such as hardware security modules, access control, and audit logging. PUFBind's architecture deliberately separates the cryptographic mechanism from its infrastructural deployment, ensuring clear security boundaries and facilitating modular implementation.

6. SECURITY ANALYSIS

This section provides a comprehensive security analysis of PUFBind. We analyze the security of the underlying tweakable block cipher construction, present the core reduction proof for device-binding security, discuss resistance against white-box key extraction attacks, and finally compare PUFBind's security properties with related work.

6.1 Security of the Underlying Tweakable Block Cipher Construction

PUFBind instantiates the CLRW2 construction with $h_1(z) = \Pi_{\text{WB}}.\text{Enc}_{K_3}(z)$ and $h_2(z) = \Pi_{\text{WB}}.\text{Enc}_{K_4}(z)$. The CLRW2 proof requires h_1, h_2 to be ϵ -AXU₂ hash functions. We first show that our instantiation satisfies $\epsilon = 2^{-128}$.

Lemma 1 (Π_{WB} as ϵ -AXU₂ Hash). *For random key K , the function $H_K(x) = \Pi_{\text{WB}}.\text{Enc}_K(x)$ satisfies for any $x \neq x'$ and Δ :*

$$\Pr[H_K(x) \oplus H_K(x') = \Delta] \leq 2^{-128} \quad (15)$$

Proof. The round function of Π_{WB} has an *SLS* structure, where S represents an S-box layer with 8-bit random permutations, and L represents a linear layer whose branch number is 5. For an 8-bit random permutation, the maximum differential probability is 2^{-4} (see [20], Section 3.1). Consequently, the maximal differential propagation probability of the round function is 2^{-20} . Since a 19-round implementation of Π_{WB} exhibits at least 7 differential active round functions (see [32], Theorem 1), the overall maximal differential propagation probability of the 19-round Π_{WB} is upper bounded by 2^{-140} , which is less than 2^{-128} . Therefore, the full 32-round Π_{WB} provides additional security margin, and we have $\epsilon \leq 2^{-128}$.

Consequently, in PUFBind, h_1 and h_2 are independent ϵ -AXU₂ hash functions with $\epsilon = 2^{-128}$. With this foundation established, we now proceed to the main security theorems.

Theorem 1 (Black-box Confidentiality of PUFBind). *If Π_{WB} is an SPRP, then PUFBind is a secure STPRP as defined in Section 4.3.1.*

Proof. PUFBind instantiates the CLRW2 construction with Π_{WB} as the underlying block cipher and $h_1(t) = \Pi_{\text{WB}}.\text{Enc}_{K_3}(t)$, $h_2(t) = \Pi_{\text{WB}}.\text{Enc}_{K_4}(t)$. By Lemma 1, h_1 and h_2 are ϵ -AXU₂ hash functions with $\epsilon = 2^{-128}$. The CLRW2 security theorem [28] states that when instantiated with an SPRP and 2^{-n} -AXU₂ hash functions, the resulting tweakable block cipher is STPRP-secure. Therefore, PUFBind achieves STPRP security. More precisely, for any adversary $\mathcal{A}$ making at most q queries to the encryption and decryption oracles, there exists an SPRP adversary $\mathcal{B}$ against Π_{WB} such that:

$$\text{Adv}_{\text{PUFBind}}^{\text{STPRP}}(\mathcal{A}) \leq 2 \cdot \text{Adv}_{\Pi_{\text{WB}}}^{\text{SPRP}}(\mathcal{B}) + \frac{6q^3\hat{\epsilon}^2}{1 - q^3\hat{\epsilon}^2} \quad (16)$$

where $\hat{\epsilon} = \max\{2^{-128}, 1/(2^{128} - 2q)\}$. Therefore, PUFBind is secure as an STPRP up to $q = 2^{85}$ queries, as proved in [28].

Consequently, PUFBind achieves both IND-CPA and IND-CCA security when deployed as an encryption scheme. From the server's perspective, this ensures that ciphertexts are computationally indistinguishable from random values, thereby safeguarding data confidentiality against passive and active eavesdroppers alike.

6.2 Device-Binding Security Reduction

The core security contribution of PUFBind is its provable device-binding property. We now present formal reductions showing that breaking PUFBind's device-binding security requires breaking the fundamental security properties of the PUF.

Theorem 2 (Device-Binding Security w.r.t NF-CA). *Let Π_{WB} be an SPRP. Let Φ be a $(q, \epsilon_{\text{sel}})$ -selectively unclonable PUF system (Definition 1). Then PUFBind is $\left(q, \epsilon_{\text{sel}} + \frac{q^2}{2^{n+1}} + \text{negl}(\lambda)\right)$ -NF-CA secure, where $\text{negl}(\lambda)$ is a negligible function in λ .*

Proof. We construct a reduction algorithm $\mathcal{B}$ that uses any efficient NF-CA adversary $\mathcal{A}$ against PUFBind to break the selective unclonability of Φ . The proof proceeds via a sequence of games, each modifying the original experiment slightly while bounding the change in the adversary's success probability.

Construction of the reduction $\mathcal{B}$:

1. $\mathcal{B}$ receives a PUF system Φ from the selective unclonability challenger.
2. $\mathcal{B}$ simulates the PUFBind environment for $\mathcal{A}$, by generating four independent master keys, compiling the corresponding white-box keys, and providing $\mathcal{A}$ with the complete white-box decryption program. Crucially, the code for the PUF Φ itself is not included in this program, only an interface to query Φ is provided.
3. After making at most q queries, $\mathcal{A}$ outputs a cloned program P' intended to run on a different physical device equipped with a distinct PUF $\Phi' \neq \Phi$.
4. $\mathcal{B}$ selects a fresh challenge x^* that was never queried by $\mathcal{A}$ during the simulation. It then queries $\Phi(x^*, \perp)$ to obtain z^* , and encrypts a uniformly random message $m \leftarrow \{0, 1\}^{128}$ using $\text{PEnc}(K, x^*, m)$ with tweak z^* to obtain ciphertext c^* .
5. $\mathcal{B}$ runs the cloned program P' on input (c^*, x^*) in an environment simulating a device with the different PUF Φ' , and obtains output $\hat{m}$.
6. If $\hat{m} = m$, $\mathcal{B}$ outputs the description of Φ' constructed by $\mathcal{A}$; otherwise, it outputs $\perp$.

Game-based analysis: We analyze the adversary's success probability through a sequence of games, tracking the probability of the event Win_i that the adversary wins in Game i .

Game G_0 : The original NF-CA experiment as defined in Section 4.3.3. Let $\Pr[\text{Win}_0]$ be the adversary's success probability in this game.

Game G_1 : Identical to G_0 , except we replace the concrete hash functions $h_1(z) = \Pi_{\text{WB}}.\text{Enc}_{K_3}(z)$ and $h_2(z) = \Pi_{\text{WB}}.\text{Enc}_{K_4}(z)$ with independent random permutations P_3, P_4 . By Lemma 1, h_1 and h_2 are 2^{-128} -AXU₂. The distinguishing advantage between ϵ -AXU₂ hash functions and random permutations is bounded by the birthday term:

$$|\Pr[\text{Win}_1] - \Pr[\text{Win}_0]| \leq \frac{q^2}{2^{n+1}} \quad (17)$$

Game G_2 : Identical to G_1 , except we replace the underlying block cipher Π_{WB} with keys K_1, K_2 with truly random permutations. Since Π_{WB} is assumed to be an SPRP:

$$|\Pr[\text{Win}_2] - \Pr[\text{Win}_1]| \leq \text{negl}_1(\lambda) \quad (18)$$

Analysis of Game G_2 : In G_2 , for $\mathcal{A}$ to win, the cloned program P' with PUF Φ' must correctly decrypt a ciphertext c^* encrypted under tweak $z^* = \Phi(x^*)$. Recall the decryption process of PUFBind. Let $z' = \Phi'(x^*)$. Successful decryption requires:

$$P_3(z') = P_3(z^*) \quad \text{and} \quad P_4(z') = P_4(z^*) \quad (19)$$

Since P_3 and P_4 are permutations, this is equivalent to $z' = z^*$, unless a collision with negligible probability happens. Thus,

$$\Pr[\text{Win}_2] \leq \Pr[\Phi'(x^*) = \Phi(x^*)] + \text{negl}_2(\lambda) \quad (20)$$

where $\text{negl}_2(\lambda)$ accounts for the negligible probability of permutation collisions.

Combining all bounds: Collecting the probability differences from all game transitions and chaining the above inequalities, and we get

$$\begin{aligned} \text{Adv}_{\mathcal{A}, \text{PUFBind}}^{\text{NF-CA}} &= \Pr[\text{Win}_0] \leq \Pr[\text{Win}_1] + \frac{q^2}{2^{n+1}} \leq \Pr[\text{Win}_2] + \text{negl}_1(\lambda) + \frac{q^2}{2^{n+1}} \\ &\leq \Pr[\Phi'(x^*) = \Phi(x^*)] + \text{negl}_2(\lambda) + \text{negl}_1(\lambda) + \frac{q^2}{2^{n+1}} \leq \epsilon_{\text{sel}} + \frac{q^2}{2^{n+1}} + \text{negl}(\lambda) \end{aligned} \quad (21)$$

This completes the reduction proof. If $\mathcal{A}$ succeeds with probability significantly greater than $\epsilon_{\text{sel}} + \frac{q^2}{2^{n+1}} + \text{negl}(\lambda)$, then $\mathcal{B}$ breaks the selective unclonability of Φ with advantage greater than ϵ_{sel} , contradicting the assumption that Φ is $(q, \epsilon_{\text{sel}})$ -selectively unclonable.

6.3 White-Box Key Extraction Resistance

Constructing a formally provable white-box cipher remains an open challenge. We therefore follow the established practice in white-box cryptography [5, 20] by designing Π_{WB} according to principles known to resist practical attacks and analyzing its resilience against known attack vectors.

Design Principles and Rationale. The white-box block cipher Π_{WB} we constructed in Section 5.2 incorporates multiple defensive layers:

- *Randomized, Instance-Specific Encodings:* The core of Π_{WB} 's white-box protection is the set of 128 unique, randomly generated 8-bit S-boxes which are represented by lookup tables $T_1, \dots, T_{128}$. These are created via a DRBG seeded by the master key, ensuring each device instance has a distinct encoding. This directly counters attacks like BGE [33] that exploit fixed, known algebraic properties of S-boxes like in AES.

- *Large State and Deep Diffusion*: The 32-round SPN structure operating on a 128-bit state ensures that any local key information is thoroughly diffused across the entire state after a few rounds. The linear layer L combines bits across the entire 32-bit word with shifts (2, 10, 18, 24) chosen for high branch number.
- *Heavy Non-Linearity*: Each round applies two layers of four 8-bit S-boxes, resulting in 256 S-box lookups per encryption/decryption. This creates a highly complex and non-linear relationship between the master key and the executable code.
- *Uniform Implementation*: The entire algorithm is implemented as a sequence of table lookups and XOR operations, with no key-dependent branching or memory accesses. This eliminates timing side-channels and simplifies resistance against dynamic analysis.

Resistance to Known Attack Vectors. Since the key-related components in the construction of PUFBind are precisely the white-box block cipher Π_{WB} , the security of Π_{WB} against known classic white-box attacks is analyzed here.

- *BGE and Affine Equivalence Attacks*: These attacks [33-34] work by finding affine mappings between the encoded implementation and a known cipher description. The use of completely random, key-derived S-boxes, rather than encoded versions of a fixed S-box as in AES, removes the underlying fixed reference, and may hinder direct application of attacks relying on a known fixed S-box. Nevertheless, dedicated cryptanalysis against this specific structure remains a subject for future investigation.
- *Algebraic Attacks*: An adversary could attempt to model the 128 random S-boxes and the linear layer as a large system of multivariate equations. However, with 128 independent random permutations over $\mathbb{F}_{2^{8n}}$, the number of possible configurations is astronomically large ($\approx (256!)^{128}$), and the derived equations are expected to be of very high degree, rendering Gröbner basis and related algebraic techniques impractical.
- *Differential/Linear Cryptanalysis*: While the random S-boxes preclude pre-computed differential/linear tables, the design inherits security margins from SPN structures akin to SMS4 [32, 35]. The 32 rounds provide a substantial security margin against statistical attacks.
- *Dynamic Analysis and Code Cloning*: As proven in Theorem 2, the cloned code alone is useless without the specific PUF. Dynamic analysis like debugging and tracing may reveal intermediate table indices, but these indices correspond to values encrypted under the PUF-derived tweak (Δ_1, Δ_2) . Without the correct PUF response on the attacker's platform, these intermediates are effectively random and do not leak key information.

In summary, while we cannot claim provable white-box security, Π_{WB} adheres to the strongest known design principles and shows robustness against all currently published white-box attacks targeting similar constructions.

6.4 Security Comparison with Related Work

Table 3 provides a qualitative comparison of PUFBind’s security properties against representative prior works in white-box cryptography and PUF-based security. The comparison highlights PUFBind’s unique position:

- *Comparison with Traditional White-Box Cryptography:* Schemes like [5, 20] focus solely on software obfuscation and provide no defense against copying the software itself. PUFBind directly addresses this via cryptographic binding to hardware.
- *Comparison with PUF-based Authentication and Encryption:* Previous PUF integrations (e.g., [2-4]) often focused on authentication or ad-hoc constructions without a full encryption specification or reduction proof for the integrated system. Especially, [3] provides a security proof only for an abstract primitive framework, but fails to prove the device-binding security when instantiating the concrete tweakable Even-Mansour construction.
- *Comparison with Theoretical Device-Binding Models:* While recent theoretical frameworks for device-binding white-box cryptography [23, 25] establish important security notions, their constructions rely on strong, currently impractical primitives like indistinguishability obfuscation (*iO*). PUFBind addresses this gap by providing a concrete, efficiently implementable solution that reduces security to standard PUF properties, making it immediately applicable to real-world IoT deployments without sacrificing provable security guarantees.

PUFBind achieves a practical balance: it provides a concrete cryptographic construction with provable device-binding security relying on standard PUF assumptions, practical white-box resistance through careful design, and efficient performance suitable for IoT devices, as demonstrated in the next section.

7. PERFORMANCE EVALUATION AND PRACTICAL DEPLOYMENT

7.1 Concrete Security Bounds and Deployment Considerations

The asymptotic bound in Theorem 2 provides formal security guarantees. To guide practical implementation, we derive concrete security estimates based on established PUF characteristics and cryptographic standards.

In the industrial sector, PUFs have gained considerable attention as a solution to the security challenges associated with IoT devices. This is particularly true for SRAM PUFs, which are intrinsic and require no hardware modifications, making them ideal for a wide range of existing electronic products that already incorporate SRAM memory. Synopsys[®] is a well-established company that has made significant contributions to the development of PUFs. By combining error correction and privacy amplification, their

[®] <https://www.synopsys.com/designware-ip/technical-bulletin/sram-puf-secure-silicon-fingerprint.html>.

Table 3. Security Properties Comparison.

Scheme	Conf.*	KER ⁺	CCR ⁺	Device-Binding Proof	Hardware Requirement
[5]	Yes	No	No	No	None
[20]	Yes	Yes	Space Hardness ⁺⁺	No	None
[3]	Yes	Yes	Yes	No	PUF
[4]	Yes	Yes	Yes	No	Ideal PUF
[23, 25]	Yes	Yes	Yes	Yes ($i\mathcal{O}$ Reduction)	Android Key Store
PUFBind	Yes	Yes	Yes	Yes (PUF Reduction)	Weak PUF

* Confidentiality. + Key Extraction Resistance. + Code Cloning Resistance. ++ “Space Hardness” means that the white-box implementation is designed to be extremely large (e.g., hundreds of megabytes or more), making it practically infeasible to extract or clone the entire code on resource-constrained IoT devices. However, this does not provide cryptographic binding to a specific hardware instance, and an adversary with sufficient storage could still clone the software.

SRAM PUFs are constructed, which create 256-bit uniformly random keys from as little as 1 KB of SRAM while ensuring complete randomness. However, such SRAM-PUFs are designed as key-generating PUFs. They produce only one static secret and therefore offer a single CRP effectively. This is insufficient for our scheme, which requires a large number of distinct challenge-output pairs.

Recent academic advances have demonstrated methods to significantly expand the CRP space of SRAM PUFs, for instance, through reconfigurable PUF structures [36], XOR-based entropy accumulation [37], or sequence-dependent addressing [31]. Following the enhanced design of Lu et al. [31], we can configure a 1 KB SRAM as a 64×128 array (rows $\times$ columns), yielding a 128-bit response per CRP. Using a sequence length $r = 5$, the theoretical CRP space reaches 1.4×10^{14} pairs, corresponding to a challenge length of approximately 47 bits, together with the bit error rate lower than 0.8%. In practice, the server does not need to store the entire theoretical CRP set. For a typical IoT device lifetime, storing 10^5 CRPs per device is sufficient. This results in a manageable server-side storage overhead of about 2 MB per device.

Our scheme consumes one CRP for each encryption, guaranteeing strict forward secrecy. With 10^5 CRPs, a device can perform 100,000 independent encryptions. Over a 10-year lifetime, this capacity comfortably covers several common low-frequency IoT scenarios, as summarized in Table 2. All listed scenarios remain within the 100,000-encryption limit and retain 12%–78% of CRPs as a safety margin for unexpected operations. Consequently, the scheme is well-suited for IoT applications that require strong forward security and have moderate encryption frequencies. Note that high-frequency applications, such as real-time video surveillance, vehicle-to-everything communication, or industrial programmable logic control, exceed this limit. To support such cases, the scheme can be extended, for example by XORing a session identifier with the CRP response to create multiple tweaks per CRP.

Assume an adversary captures a device for one month and observes its runtime behavior during at most 1000 decryption operations via debug interfaces or side-channel probes thereby obtaining 1000 pairs of (x, z) , i.e., $q \leq 1000$. Based on the 80% min-entropy property of the SRAM PUF and the secure design of the fuzzy extractor [27], we adopt the conservative parameter $\epsilon_{\text{sel}} \leq 2^{-80}$. Even when the adversary obtains $q = 1000$ CRPs, this parameter remains valid due to the response independence of weak PUFs. Substituting this parameter into Theorem 2 gives an NF-CA security bound of approximately 2^{-80} , indicating that an adversary's success probability in a single decryption attempt does not exceed 2^{-80} .

7.2 Experimental Design and Testbed Configuration

To validate the practical viability of PUFBind in real-world IoT scenarios, we conducted a comprehensive performance evaluation on representative hardware platforms. Our experiments were designed to answer two key questions:

1. *Is PUFBind efficient enough for resource-constrained IoT devices?*
2. *How does it compare with state-of-the-art white-box encryption schemes that do not support hardware binding?*

We selected four hardware platforms covering both server-class and embedded-class devices, as summarized in Table 4. The workstation (Intel i7-12700F) and laptop (Apple M2) simulate cloud servers that perform encryption. The Raspberry Pi 5 (ARM Cortex-A76) and Xilinx Zynq-7010 (ARM Cortex-A9) represent typical IoT endpoints that execute decryption. This selection ensures that our evaluation spans the architectural diversity of actual IoT deployments, from high-performance servers to severely constrained edge devices.

Table 4. Configurations of Testing Platforms.

Device	Architecture	Processor	RAM	OS
Workstation	x86_64	Intel i7-12700F@2.1GHz	16 GB DDR4	Ubuntu 24.04.1 LTS
Laptop	ARM64	Apple M2@3.5GHz	16 GB LPDDR5	MacOS Sequoia 15.5
Raspberry Pi 5	ARMv8	ARM Cortex-A76@2.4GHz	8 GB LPDDR4X SDRAM	Linux raspberrypi 6.12.25
Xilinx Zynq-7010	ARMv7	ARM Cortex-A9@666MHz	512 MB DDR3	Linux zynq 4.9.0-xilinx

All implementations were coded in C and compiled with GCC under -O3 optimization. We measured execution time, throughput, clock cycles, and cycles per byte (CPB). Each experiment was repeated 100,000 times to ensure statistical significance, and results were averaged.

7.3 Performance of PUFBind's Building Block

Table 5 presents the detailed performance of PUFBind's core components. The white-box block cipher Π_{WB} , which serves as the cryptographic engine, shows efficient key generation (0.45 μ s) and compilation (1 ms). Its encryption and decryption throughput on the laptop reach 136.56 MB/s and 141.92 MB/s, respectively, demonstrating that the white-box design does not introduce prohibitive overhead.

Table 5. Overall Performance Evaluation of Π_{WB} and PUFBind[†].

Algorithm		TC*	Experimental Performance				Storage
			Time	Throughput	Cycles	CPB	
Π_{WB}	KGen	$\mathcal{O}(1)$	0.45 μs	33.91 MB/s	1575	98	128 bit
	KCom	$\mathcal{O}(1)$	1 ms	124.75 MB/s	3.5×10^6	27	32 KB
	Enc	$\mathcal{O}(n)$	14.30 μs	136.56 MB/s	50060	24	32 KB
	Dec	$\mathcal{O}(n)$	13.76 μs	141.92 MB/s	48169	24	32 KB
PUFBind	PKGen	$\mathcal{O}(1)$	0.69 μs	88.46 MB/s	2415	38	512 bit
	PKCom	$\mathcal{O}(1)$	2.99 ms	41.82 MB/s	1.05×10^7	80	128 KB
	PEnc	$\mathcal{O}(n)$	30.56 μs	63.90 MB/s	106976	52	2 MB
	PDec	$\mathcal{O}(n)$	28.85 μs	67.70 MB/s	100978	49	128 KB

[†] All tests are under the laptop. KGen, KCom, PKGen, and PKCom are tested once. Enc, Dec, PEnc, and PDec are measured by taking the average over 100, 000 repetitions, with each repetition involving the encryption (resp. decryption) of a random 2 KB message.

* Time Complexity.

For the full PUFBind scheme, the encryption and decryption throughputs on the laptop are 63.90 MB/s and 67.70 MB/s, respectively. This represents a reduction of approximately 53.2% for encryption (from 136.56 MB/s) and 52.3% for decryption (from 141.92 MB/s) compared to the standalone Π_{WB} . Considering that PUFBind invokes four instances of Π_{WB} (two for core encryption/decryption and two for tweak generation) and incorporates additional XOR operations for hardware binding, this acceptable performance overhead confirms that the security benefits of provable anti-cloning are achieved at a practical computational cost. The near-symmetric throughput of encryption and decryption also validates the balanced design of the construction.

Figure 5 and Figure 6 illustrate how encryption and decryption performance scale with message length. As expected, throughput increases with larger messages due to the amortization of fixed overheads (e.g., PUF query, tweak computation). However, the relationship is log-linear rather than purely linear, reflecting caching effects in modern CPU architectures. The figures also reveal notable performance

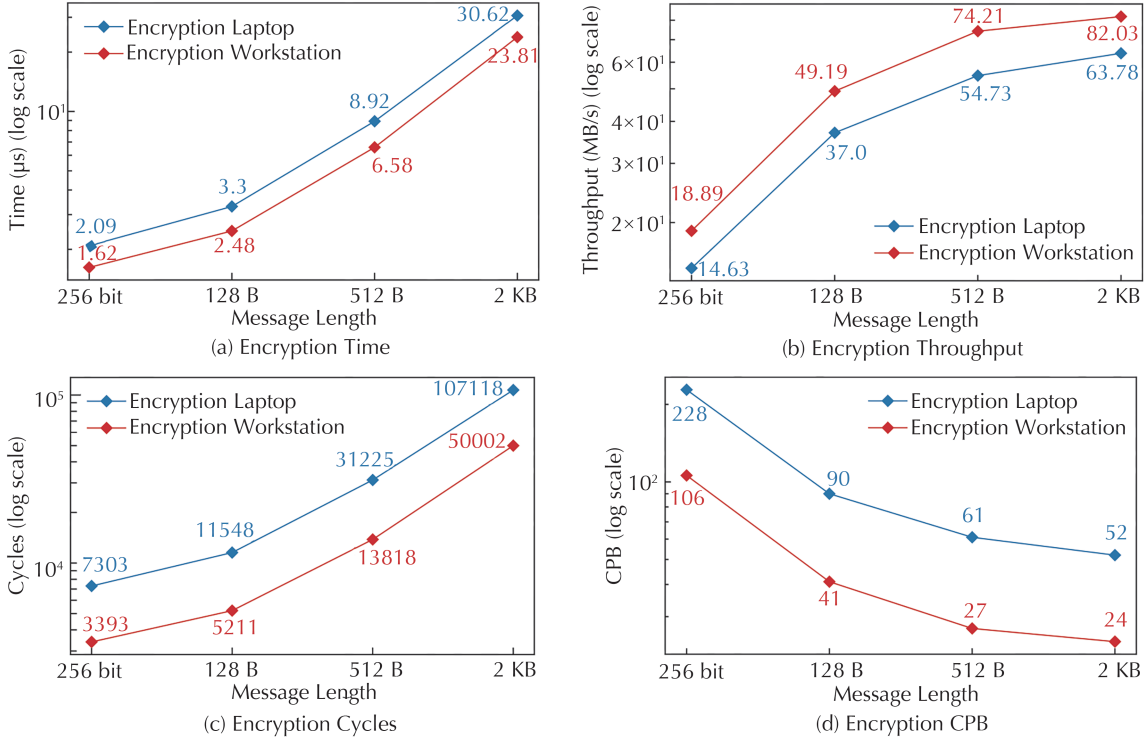


Figure 5. Encryption Performance of PUFBind.

differences across hardware platforms, underscoring PUFBind’s scalability. For encryption, the workstation demonstrates a clear advantage over the laptop. At a typical 2 KB message size, the encryption CPB on the workstation is approximately half of that on the laptop, highlighting the benefit of the workstation’s higher memory bandwidth and core count for server-side operations. Conversely, for decryption which is performed on the IoT endpoint, the Raspberry Pi 5 significantly outperforms the resource-constrained Xilinx Zynq-7010. Specifically, the decryption time on the Raspberry Pi 5 is about 20 times faster than on the Zynq-7010 for the same message length. This stark contrast validates the practical efficiency of PUFBind’s white-box decryption routine on more capable, modern ARM-based edge platforms while also defining its performance baseline on severely constrained hardware.

7.4 Comparison with Existing White-Box Schemes

To contextualize PUFBind’s performance, we compared it against three prominent white-box encryption schemes that lack hardware-binding protections: SPNbox-16 [20], WEM [38], and WARX [15]. The results, summarized in Figure 7, show that PUFBind achieves superior efficiency in both encryption and decryption. On the workstation, PUFBind encrypts 2 KB messages 28× faster than SPNbox-16 (in terms of execution time), while on the Raspberry Pi 5, it decrypts 23× faster. This significant speedup stems from two factors:

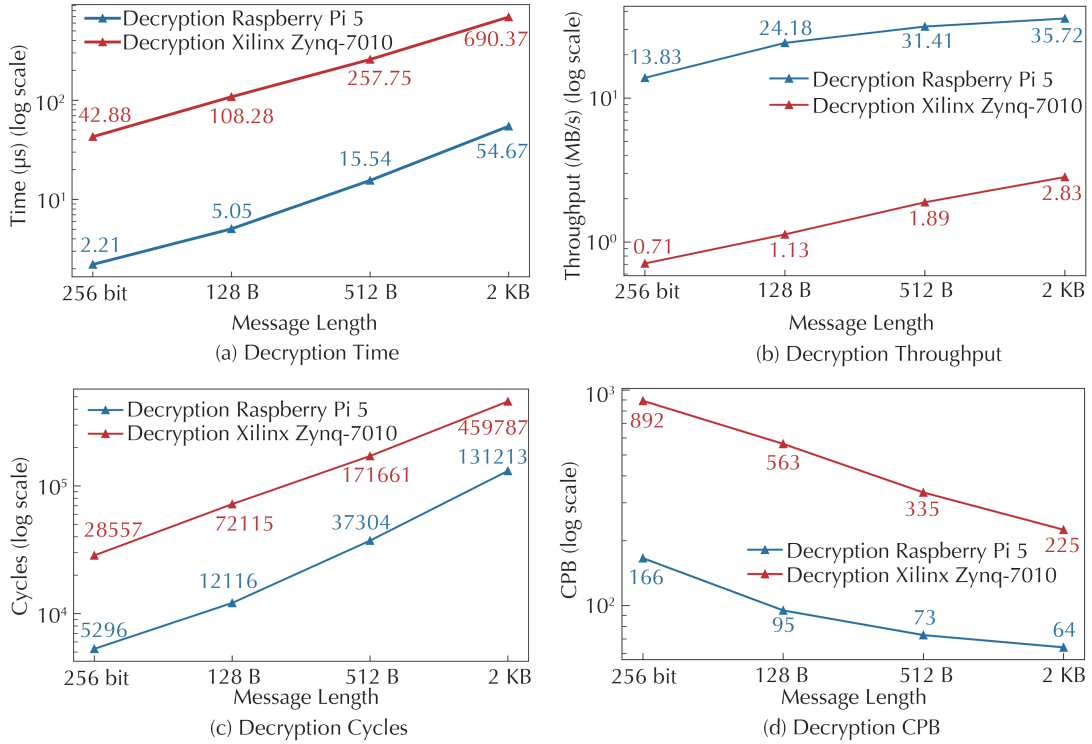


Figure 6. Decryption Performance of PUFBind.

1. *Lightweight tweak integration:* PUFBind's use of the CLRW2 construction adds minimal overhead beyond the core block cipher.
2. *Optimized white-box design:* Π_{wb} employs a streamlined SPN structure with 8-bit S-boxes, reducing table lookups and memory accesses compared to more complex designs.

Crucially, this performance advantage does not come at the expense of security. Unlike the compared schemes, PUFBind provides provable anti-cloning security through PUF binding, which is a feature none of the other schemes offer.

7.5 Storage and Memory Footprint

PUFBind's white-box decryption routine requires 128 KB of static memory for four sets of lookup tables. On the device side, only the 128 KB tables must reside in persistent storage, which is a reasonable demand for many IoT microcontrollers equipped with flash memory.

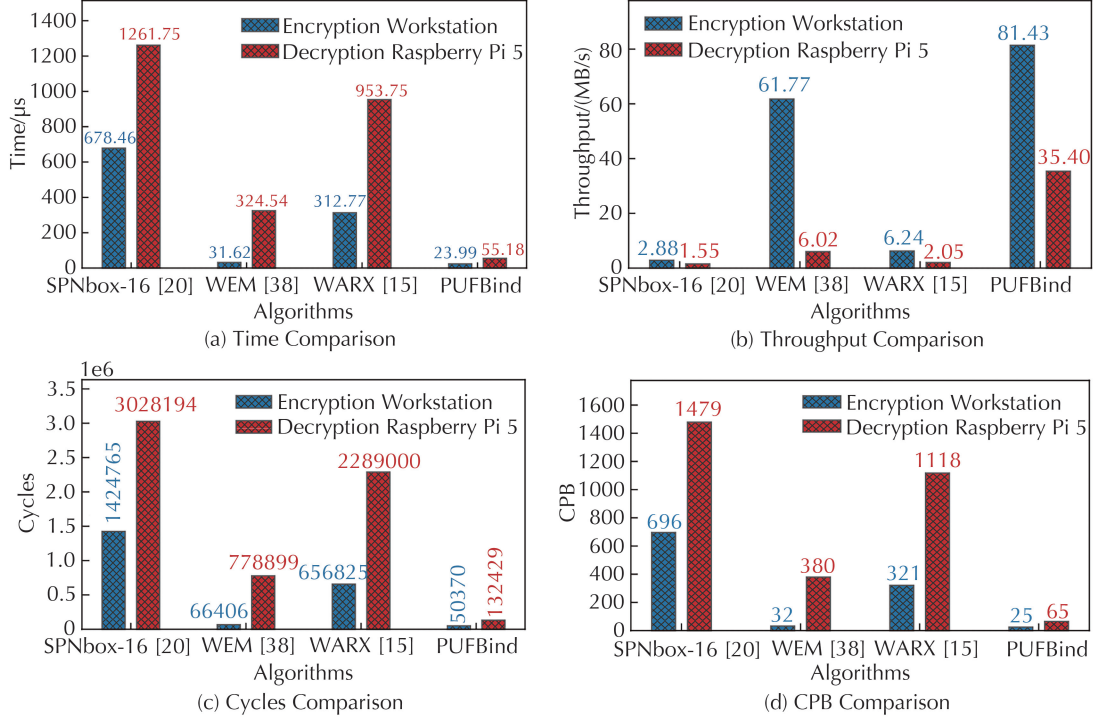


Figure 7. Performance Comparison.

Baseline scheme for small-to-medium scale deployments. To support forward secrecy, the server stores a pre-generated set of challenge-output pairs (x, z) for each device. A conservative configuration with 10^5 CRPs consumes approximately 2 MB of storage per device. As shown in Table 2, this capacity comfortably covers typical IoT scenarios such as smart meters or insulin pumps over a 10-year lifetime, and the per-device storage overhead is acceptable for small-to-medium scale deployments.

Alternative optimizations for large-scale deployments. For cloud platforms managing millions of devices, the aggregate storage under the baseline scheme would scale to terabytes, which warrants further optimization. We therefore propose two alternative optimizations that eliminate the need for per-device CRP storage, making the scheme practical at million-device scale.

- *CRP derivation from a master seed.* Instead of storing all CRPs explicitly, the server stores a single master seed per device and derives CRPs on-the-fly using a pseudorandom function (PRF). Each encryption then requires one PRF evaluation to generate the required CRP, which is negligible for a cloud server. This reduces per-device storage from 2 MB to a single seed, e.g., 32 bytes. As a result, the scheme scales gracefully to millions of devices without imposing a significant storage burden. This optimization introduces a single point of failure (the master seed), which must be protected with equivalent security strength as the CRPs themselves.

- *Tiered storage.* For deployments that prefer to avoid online PRF evaluation, the server can adopt a hybrid approach: generate and store CRPs in batches (e.g., the next 10^4 CRPs per device) in fast storage, while keeping the master seed as backup. Once a batch is exhausted, the next batch is generated from the seed and cached. This limits active storage to a small constant per device (e.g., 220 KB) while retaining the efficiency of precomputed CRPs.

Both optimizations eliminate the 2 MB per-device storage overhead of the baseline scheme, enabling PUFBind to scale gracefully to millions of IoT devices.

7.6 Discussion: Balancing Performance and Security

The experimental results demonstrate that PUFBind successfully balances strong security with practical efficiency. Its encryption/decryption speeds are sufficient for a wide range of IoT applications, from smart meters (one transmission per hour) to insulin pumps (multiple commands per day), as quantified in Table 2. The added computational cost of PUF binding, compared to software-only white-box schemes, is acceptable and justified by the fundamental protection it provides against device cloning.

Furthermore, the scheme's forward-secrecy mechanism enabled by one-time use of PUF challenges imposes no additional runtime penalty, as challenges are merely looked up from a pre-computed database. This makes PUFBind particularly attractive for long-lived deployments where periodic key renewal is impractical.

7.7 Limitations and Future Optimizations

While PUFBind performs well on general-purpose CPUs, its current implementation does not exploit hardware acceleration features such as AES-NI or ARM Cryptography Extensions. Future work could explore hybrid designs where symmetric-key operations are offloaded to dedicated hardware instructions, further improving throughput. Additionally, the storage overhead of PUF challenge-output pairs could be reduced by employing PUF designs with larger entropy density or by combining PUF responses with session identifiers to reuse challenges.

Beyond performance, we acknowledge three core limitations. First, our white-box security analysis is heuristic, following established design principles but lacking formal proof. Second, we have not performed experimental validation against physical side-channel attacks. Third, our implementation does not integrate a physical PUF or fuzzy extractor, and real-world deployments will incur additional hardware latency and error-correction overhead. These limitations define the boundary of our conclusions: the provable security guarantees for server-side confidentiality and device binding remain formally valid, while end-to-end physical security additionally depends on proper PUF integration, side-channel protections, and continued resilience to white-box cryptanalysis.

8. CONCLUSION

This paper proposes PUFBind, a unified cryptographic solution for IoT white-box threats by integrating PUF-based hardware binding with tweakable encryption. It achieves provable security against code cloning attacks while maintaining practical efficiency on constrained devices. The design closes the gap between hardware-assisted security and full encryption specifications, offering a practical design toward deployment for physically exposed IoT endpoints. Future work may explore dynamic challenge management and extensions for high-frequency communication scenarios.

AUTHOR CONTRIBUTIONS

Jun Liu: Conceptualized the research framework, designed the white-box cipher and PUFBind construction, and completed experimental validation and manuscript drafting;

Yang Shi: Analyzed security of the construction, and ensured the accuracy and completeness of the manuscript;

Jie Chen: Supervised research feasibility, and provided experiment platforms;

Xiaoli Dong: Evaluated model performance and analyzed results;

Lu Zhou: Supported the interpretation and validation of the results, reviewed and edited the manuscript to maintain high-quality research standards;

Bo Yang: Provided guidance and support throughout the study, oversaw the research project and provided strategic direction.

ACKNOWLEDGEMENTS

This work was supported in part by the National Natural Science Foundation of China [62302285, 62472150, 62472318, 62172301, U2001205]; Young Talent Fund of Association for Science and Technology in Shaanxi of China [20240124]; Open Funding of State Key Laboratory of Integrated Services Networks [ISN26-11], and Key Research and Development Program of Shaanxi [2023-YBGY-015]. We appreciate reviewers for their insightful comments.

REFERENCES

- [1] Y. Shi, W. Wei, F. Zhang, X. Luo, Z. He, and H. Fan, "SDSRS: A novel white-box cryptography scheme for securing embedded devices in IIoT," *IEEE Trans. Ind. Inform.*, vol. 16, no. 3, pp. 1602–1616, Mar. 2020.
- [2] B. Kim, S. Yoon, Y. Kang, and D. Choi, "Secure IoT device authentication scheme using key hiding technology," in *2020 Int. Conf. Inf. Commun. Technol. Converge.*, 2020, pp. 1808–1810.

- [3] S. Rasoamiaramanana, M. Minier, and G. Macario-Rat, "A white-box encryption scheme using physically unclonable functions," in *Proc. 17th Int. Joint Conf. e-Bus. and Telecommun.*, 2020, pp. 279–286.
- [4] T. Asanuma and T. Isobe, "Even-Mansour space-hard cipher: White-box cryptography cipher meets physically unclonable function," *J. Inf. Process.*, vol. 31, pp. 88–96, Jan. 2023.
- [5] S. Chow, P. Eisen, H. Johnson, and P. C. V. Oorschot, "White-box cryptography and an AES implementation," in *Proc. 9th Int. Workshop Sel. Areas Cryptography*, 2002, pp. 250–270.
- [6] L. Zhou, C. Su, Y. Wen, W. Li, and Z. Gong, "Towards practical white-box lightweight block cipher implementations for IoTs," *Future Gener. Comput. Syst.*, vol. 86, pp. 507–514, Sep. 2018.
- [7] C. Jin, Z. Zeng, W. Miao, Z. Bao, and R. Zhang, "A nonlinear white-box SM4 implementation applied to edge IoT agents," in *2021 IEEE 5th Conf. Energy Internet Energy Syst. Integration*, 2021, pp. 3358–3363.
- [8] W. Miao, C. Jin, Z. Zeng, Z. Bao, X. Wei, and R. Zhang, "A white-box SM4 implementation by introducing pseudo states applied to edge IoT agents," in *2022 4th Asia Energy Elect. Eng. Symp.*, 2022, pp. 154–160.
- [9] T. Ma, "White-box Schnorr signature for Internet of Things security," in *2020 5th Int. Conf. Mech., Control Comput. Eng.*, 2020, pp. 1939–1942.
- [10] A. O. Bang, U. P. Rao, P. Raj, and A. Kumar, "White-box encryption scheme for resource-constrained IoT devices," in *Sel. Proc. 1st Int. Conf. Secur., Privacy Data Analytics*, 2021, pp. 257–268.
- [11] A. O. Bang and U. P. Rao, "Design and evaluation of a novel white-box encryption scheme for resource-constrained IoT devices," *J. Supercomputing*, vol. 78, pp. 11111–11137, Feb. 2022.
- [12] A. Saha and C. Srinivasan, "White-box cryptography based data encryption-decryption scheme for IoT environment," in *2019 5th Int. Conf. Adv. Comput. Commun. Syst.*, 2019, pp. 637–641.
- [13] L. Marin, "White box implementations using non-commutative cryptography," *Sensors*, vol. 19, no. 5, Mar. 2019, Art. no. 1122.
- [14] B. Wang, S. Chen, J. Chen, and X. Wang, "DWB-AES: An implementation of dynamic white-box based on AES," *J. Commun.*, vol. 42, no. 2, pp. 177–186, Feb. 2021.
- [15] J. Liu, V. Rijmen, Y. Hu, J. Chen, and B. Wang, "WARX: Efficient white-box block cipher based on ARX primitives and random MDS matrix," *Sci. China-Inf. Sci.*, vol. 65, no. 3, Aug. 2022, Art. no. 132302.
- [16] J. Cho, K. Y. Choi, O. Dunkelman, N. Keller, D. Moon, and A. Vaidberg, "Hybrid WBC: Secure and efficient white-box encryption schemes," in *15th Int. Conf. Cryptology Netw. Secur.*, 2016, pp. 749–754.
- [17] Y. Shi, W. Wei, Z. He, and H. Fan, "An ultra-lightweight white-box encryption scheme for securing resource-constrained IoT devices," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 16–29.
- [18] Y. Shi, X. Wang, and H. Fan, "Light-weight white-box encryption scheme with random padding for wearable consumer electronic devices," *IEEE Trans. Consum. Electron.*, vol. 63, no. 1, pp. 44–52, Feb. 2017.
- [19] Y. Shi, W. Wei, H. Fan, M. H. Au, and X. Luo, "A light-weight white-box encryption scheme for securing distributed embedded devices," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1411–1427, Oct. 2019.
- [20] A. Bogdanov, T. Isobe, and E. Tischhauser, "Towards practical whitebox cryptography: Optimizing efficiency and space hardness," in *Proc. 22nd Int. Conf. Theory Appl. Cryptology Inf. Secur.*, 2016, pp. 126–158.
- [21] D. G. V. Albricci, M. Ceria, F. Cioschi, N. Fornari, A. Shakiba, and A. Visconti, "Measuring performances of a white-box approach in the IoT context," *Symmetry*, vol. 11, no. 8, Aug. 2019, Art. no. 1000.
- [22] R. Shiba, R. Anand, K. Minematsu, and T. Isobe, "Cubicle: A family of space-hard ciphers for IoT," *IET Inf. Secur.*, vol. 17, no. 1, pp. 131–146, Jul. 2023.
- [23] E. A. Bock, C. Brzuska, M. Fischlin, C. Janson, and W. Michiels, "Security reductions for white-box key-storage in mobile payments," in *Proc. 26th Int Conf. Theory Appl. Cryptology Inf. Secur.*, 2020, pp. 221–252.

- [24] E. A. Bock, A. Amadori, C. Brzuska, and W. Michiels, "On the security goals of white-box cryptography," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 2, pp. 327–357, Mar. 2020.
- [25] S. Agrawal, E. A. Bock, Y. Chen, and G. J. Watson, "White-box cryptography with global device binding from message-recoverable signatures and token-based obfuscation," in *Proc. 14th Int. Workshop Constructive Side-Channel Anal. Secur. Des.*, 2023, pp. 241–261.
- [26] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, Mar. 2008.
- [27] F. Armknecht, R. Maes, A. R. Sadeghi, F. X. Standaert, and C. Wachsmann, "A formal foundation for the security features of physical functions," in *2011 IEEE Symp. Secur. Privacy*, 2011, pp. 397–412.
- [28] W. Landecker, T. Shrimpton, and R. S. Terashima, "Tweakable blockciphers with beyond birthday-bound security," in *Proc. 32nd Annu. Cryptology Conf.*, 2012, pp. 14–30.
- [29] *Recommendation for random number generation using deterministic random bit generators (revised)*, NIST Standard 800–90A, 2015.
- [30] A. Bacher, O. Bodini, H. K. Hwang, and T. H. Tsai, "Generating random permutations by coin tossing: Classical algorithms, new analysis, and modern implementation," *ACM Trans. Algorithms*, vol. 13, no. 2, Apr. 2017, Art. no. 24.
- [31] L. Lu and T. T-H. Kim, "A high reliable SRAM-based PUF with enhanced challenge-response space," *IEEE Trans. Circuit Syst. II: Exp. Briefs*, vol. 69, no. 2, pp. 589–593, Feb. 2022.
- [32] N. Wang and Q. Yin, "Security evaluation for SMS4-typed ciphers structure against differential and linear cryptanalysis," *J. Cryptologic Res.*, vol. 2, no. 2, pp. 189–196, Apr. 2015.
- [33] O. Billet, H. Gilbert, and C. Ech-Chatbi, "Cryptanalysis of a white box AES implementation," in *Int. Conf. Sel. Areas Cryptography*, 2004, pp. 227–240.
- [34] W. Michiels, P. Gorissen, and H. D. L. Hollmann, "Cryptanalysis of a generic class of white-box implementations," in *Int. Conf. Sel. Areas Cryptography*, 2008, pp. 414–428.
- [35] B. Zhang and C. Jin, "Practical security against linear cryptanalysis for SMS4-like ciphers with SP round function," *Sci. China-Inf. Sci.*, vol. 55, no. 9, pp. 2161–2170, Sep. 2012.
- [36] S. Baek, G. H. Yu, J. Kim, C. T. Ngo, J. K. Eshraghian, and J-P. Hong, "A reconfigurable SRAM based CMOS PUF with challenge to response pairs," *IEEE Access*, vol. 9, pp. 79947–79960, May 2021.
- [37] J. Garrard, M A. Rios, and B. Cambou, "Enhancing the SRAM PUF with an XOR gate," *Appl. Sci.*, vol. 14, no. 21, Nov. 2024, Art. no. 10026.
- [38] J. Cho *et al.*, "WEM: A new family of white-box block ciphers based on the Even-Mansour construction," in *Cryptographers' Track RSA Conf.*, 2017, pp. 293–308.