Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Security and Challenges in IoT Devices

[1]Saurabh Vishwakarma, [2]Prashant Kumar Yadav
[1,2]Department of Computer Science and Engineering, VBSPU, Jaunpur, India
[1]Email : vishwakarmasaurabh2002@gmail.com
[2]Email: prashant.yadaw@gmail.com

ARTICLE INFO

Received: 17 Sep 2024

Revised: 28 Nov 2024

Accepted: 20 Dec 2024

ABSTRACT

The security of resource-constrained IoT devices hinges on robust key-management schemes that balance strong protection. In this paper, we propose and evaluate a hybrid architecture that combines a hardware root of trust provided by a Trusted Platform Module (TPM) with the ultra-fast, secure BLAKE3 hash function as a key-derivation primitive. Each device's unique endorsement key (EK) and platform measurements are sealed within its TPM, ensuring that private key material never leaves secure hardware. At runtime, a device and server perform a mutually authenticated handshake: the TPM attests to device integrity, and both parties derive session keys via BLAKE3-based HKDF using TPM-protected secrets and nonces. We implement our scheme on a representative ARM-based microcontroller platform and measure end-to-end key- establishment latency, energy consumption, and resilience to common IoT attacks (replay, man- in-the-middle, and device impersonation). Our results show that TPM-backed attestation adds less than 15 ms of overhead, while BLAKE3-driven key derivation completes in under 1 ms and requires only 12 kB of RAM—demonstrating that strong, hardware-anchored key management is feasible even on severely constrained devices. We conclude that the integration of TPM attestation with BLAKE3 KDF offers a scalable, forward-secure foundation for next-generation IoT deployments.

Keyword: Internet of Things (IoT); Key Management; Trusted Platform Module (TPM); BLAKE3; Key Derivation Function (KDF); Hardware Root of Trust

1 Introduction

The Internet of Things (IoT) has ushered in an era of pervasive connectivity, embedding smart sensors and actuators into virtually every aspect of daily life—from wearables and home automation to industrial control systems and critical infrastructure. However, this explosion of interconnected "things" also massively expands the potential attack surface, exposing sensitive data and control channels to a host of new threats. Unlike traditional computing platforms, many IoT devices operate under severe resource constraints (CPU, memory, power) and may lack built- in support for strong encryption, authenticated boot, or secure firmware updates.

As a result, common security primitives—secure key storage, mutual authentication, confidentiality, and integrity—become difficult to implement at scale. Device heterogeneity (different vendors, processors, operating systems) further fragments the ecosystem, making unified security policies and over-the-air patching problematic. In this hostile landscape, adversaries can exploit weak default credentials, unencrypted communication, outdated firmware, or even hardware trojans to mount data-theft, denial-of-service, man-in-the-middle, and supply-chain attacks. Addressing these

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

challenges requires lightweight, scalable key-management schemes, robust hardware root-of-trust anchors, and cross-layer defense mechanisms tailored to the unique constraints of IoT deployments.

The paper is structured as follows: Section 1 delivers a concise overview of key management in IoT devices, detailing the processes of cryptographic key generation, secure storage mechanisms, and ongoing key management during communication sessions.

2 Cryptographic Key Management

Key management involves four primary tasks: analyzing, assigning, creating, and distributing keys. During the key analysis process, the keying requirements are assessed to ascertain how many keys are necessary for the network and how many are needed for each node. Furthermore, an analysis may be conducted to identify which keys need to be updated. The suggested method shows improved performance on devices with limited resources. A decentralized structure enhances scalability and reduces the burden on devices.[1]

2.1 Analysis of Keys

Key length (measured in bits) determines the size of cryptographic keys used in algorithms such as AES- 128, LED, RECTANGLE, and PRINCE—all of which employ 128-bit keys. While AES supports multiple key sizes (128/192/256 bits), lightweight ciphers like LED and RECTANGLE optimize for resource- constrained IoT devices by combining 128-bit keys with energy-efficient operations. Similarly, PRINCE's 128-bit key and low-latency design enable fast encryption in real-time systems. In contrast, asymmetric algorithms like RSA-3072 (3072-bit keys) or ECC-256 (256-bit keys) prioritize mathematical complexity for robust security. However, the computational overhead of longer keys (e.g., RSA vs. AES-128) highlights the trade-off between cryptographic strength and device capabilities, necessitating context-driven choices like ChaCha20 for speed or EdDSA for compact signatures.

In our cryptographic protocol, the intentional selection of a 128-bit key for symmetric cryptography and a 256-bit key for ECC is a result of a thoughtful balancing of several factors. Although it is widely understood that increased key sizes enhance the security of protocols, we need to take into account the related costs in terms of computation and processing time.

2.2 Assignment of Key

The proposed key management has two basic characteristics: (i) the pre-distribution of keys and

(ii) the use of partial (half) keys rather than full keys.

2.2.1 Pre-Distribution of Keys

Mechanism: Cryptographic keys are embedded into devices during manufacturing or network initialization, eliminating the need for dynamic key exchange in operationally fragile environments.

Implementation:

Deterministic Key Allocation: Keys are pre-computed using device-specific identifiers (e.g., hardware fingerprints) and stored in secure enclaves (e.g., TPM/HSM modules).

Scalable Key Pools: A subset of keys from a preloaded pool is assigned to each node, enabling pairwise communication via shared secrets (e.g., Q-composite schemes).

Advantages:

Reduces communication overhead for key establishment.

Supports plug-and-play modular deployment in dynamic IoT networks.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2.2.2 Partial Key Utilization

Design: Full cryptographic keys are decomposed into fractional components (e.g., half-keys) that require collaborative reconstruction.

Methods:

Split-Key Cryptography: A full key

K is divided into K1 and K2 , where operations (e.g., encryption) demand both fragments.

Threshold-Based Reconstruction: Leverages lightweight Shamir's Secret Sharing (SSS) or polynomial-based schemes to split keys into n shares, requiring t shares ($t \leq nt \leq n$) for activation.

IoT-Specific Protocols:

ECC-Based Half-Key Exchange: Devices exchange partial elliptic curve public keys (e.g., ECDH fragments) to derive a shared secret.

Lightweight Ramp Schemes: Minimize storage by allowing partial key recovery with fewer shares, ideal for memory-limited nodes.

2.3 Generation of Key

Cryptographic key generation involves creating two distinct key categories: communication keys for secure data exchange and administrative keys for system oversight. Communication keys are dynamically generated through collaborative processes between nodes, while administrative keys are provisioned either once (static) or periodically (dynamic) over the network's operational lifespan.

2.3.1 Partial Keys

Partial Key Sets Node X :

$X = \{ P_{KX} , P_{KXi+1} , PKXi+2 , PKXi+4...\}$

Node Y:

$Y= \{P_{KYi},P_{KYi+1} , P_{KYI+2},P_{KYi+3} , P_{KYi+4}.....\}$

Order Lists

Node X sends its key sequence order to Node Y:

$X \rightarrow Y \quad : \{3,2,0,1,4,...,n-1\}$

Node Y sends its key sequence order to Node X:

$Y \rightarrow X: \{0,2,4,1,3,...,n-1\}$

Full Key Series

Combining partial keys based on Node X's order list:

$$\{P_{KXi+3}, P_{KXi} , P_{KXi+2} , P_{KXi+3} , P_{KXi+4} ,P_{KXi+3}...\}$$

2.4 Secure Key Distribution in IoT Networks

A robust key distribution framework is essential to enable secure communication among IoT nodes. This process involves securely deploying cryptographic keys to designated devices after their generation and assignment. Unlike administrative keys, which are often static, communication keys are dynamically shared post-network deployment, designed for short-term use, and periodically refreshed through phases of analysis, regeneration, and reallocation.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3 Issues and Challenges on Key Management in IoT

Secure provisioning and bootstrapping: Establishing the first trust on an IoT device – injecting its initial keys or certificates – is difficult at scale. Many IoT systems rely on factory-programmed keys or one-time passwords built into the device. For example, LPWAN standards show that nodes "all rely on pre-provisioned credentials" (such as an AppKey or SIM) to authenticate when joining a network [2] . If these initial credentials are leaked or cloned, an attacker can easily spoof or impersonate devices. Managing secure provisioning typically requires secure facilities or dedicated infrastructure (secure "burn-in" or Over-The-Air commissioning), which can be costly. Moreover, devices often ship without human interfaces or screens, so installing new keys post- manufacture is nontrivial. In sum, bootstrapping trust (e.g. via a hardware root-of-trust or secure enrollment protocol) is a major challenge in IoT key management. [2]

Physical and network attack surface: IoT devices are often deployed in unprotected environments and communicate over wireless links. An adversary may physically capture a device and extract stored keys, or intercept messages during key exchange. These factors mean key management must assume keys will be exposed and include mitigation (secure elements, frequent rekeying, tamper detection, etc.). Surveys point out that IoT's ubiquity and connectivity create many attack vectors, making it hard to maintain a consistent security framework [3].

For example, unpredictable interactions between devices ("unplanned communication") can cause devices to share keys or data with strangers if not tightly controlled [3].

Protecting keys in transit and at rest – for instance with hardware binding or trusted execution – is thus a general challenge in IoT.

Key lifecycle management: Cryptographic keys need regular maintenance: they must be renewed, rotated, or revoked when devices are lost or compromised. In IoT, this is especially hard because many devices sleep or have intermittent connectivity. A key revocation broadcast may never reach a device that is offline. One survey highlights that IoT key management must support a dynamic network: "efficient key renewal, new node addition, and key revocation" are essential as devices join or leave unexpectedly[3]. Without robust lifecycle support, an attacker who breaks a single device's key can continue to exploit it indefinitely, or stale keys may remain valid for too long. Designing automated rekeying (for example via group or broadcast keys with forward secrecy) and handling long-lived offline devices is a critical open issue.

Blockchain-based key management: Distributed ledger (blockchain) solutions have been proposed for IoT identity and key management to avoid a single point of trust. In a blockchain- based scheme, device public keys or access credentials can be stored in a tamper-resistant ledger, and smart contracts can govern issuance and revocation. This can improve auditability and avoid a central CA. However, classic blockchains introduce new challenges for IoT. Public blockchains have high overhead: every transaction (e.g. issuing a key) must be broadcast and confirmed, which is slow and energy-intensive. Each IoT node would need to run (or rely on) a full or light node, which adds storage and compute demands. Even a permissioned blockchain has consensus and replication costs. One study of a blockchain-IoT key scheme found that using public/private key pairs "necessitates more storage and processing power for IoT" [3]. In practice, many blockchain proposals for IoT offload the ledger to gateways or use simplified distributed ledgers. But issues remain: keys on a public ledger are visible to others (raising privacy concerns), and revoking or rolling back a blockchain entry is difficult. In short, while blockchain can decentralize trust, it often conflicts with IoT resource limits and real-time needs.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4 Key Management using Black3

Blake3 is a modern cryptographic hash function designed for speed, security, and versatility. It is the successor to BLAKE2 and part of the BLAKE family of hash algorithms, which were finalists in the NIST SHA-3 competition. BLAKE3 is optimized for performance across a wide range of platforms, including resource-constrained IoT devices, while maintaining strong security guarantees.

4.1Working of Blake3

- A binary tree format that allows for unlimited parallel processing.
- A reduced number of rounds in the BLAKE2s compression function (7 instead of 10).
- It supports three operational modes: hashing, keyed hashing, and key derivation. These modes simplify the API by replacing the BLAKE2 parameter block.
- Zero-cost keyed hashing, utilizing the space that was previously used by the parameter block for the key.
- An integrated extendable output (also known as XOF), which supports parallel processing and seeking, similar to BLAKE2X but differing from SHA-3 or HKDF.

5 Trusted Platform Module (TPM) for IoT Security and Key Management

The Trusted Platform Module (TPM) serves as a hardware-based root-of-trust established by the Trusted Computing Group (TCG). It is a discrete or firmware-based microcontroller on an IoT device that provides secure generation, storage, and usage of cryptographic keys. TPM chips incorporate physical tamper-resistance, and all private keys (such as the device's endorsement key or storage root key) are generated and held inside the TPM and never exposed to the host CPU or memory.

For instance, AWS states that a TPM is a specialized crypto-processor designed to ensure the device initiates in a secure and reliable state, offering hardware-based roots of trust for the integrity of the platform.

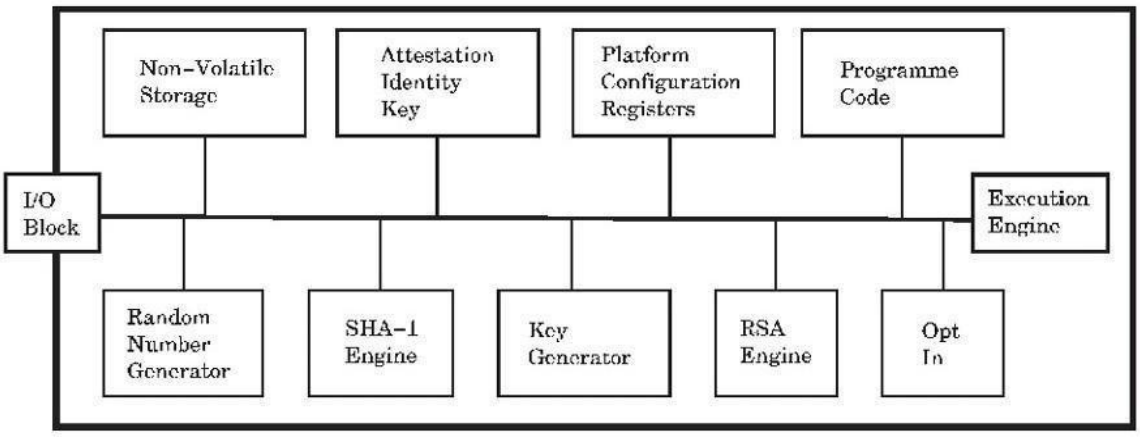

Figure 1 (Trusted Platform Module Architecture) [3]

Because the TPM runs its own isolated firmware, an attacker who compromises the operating system cannot extract its secrets.

In IoT devices, a TPM thus enables hardware-backed identity, authentication, and integrity checks that greatly exceed what software-only solutions can offer.

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5.1 TPM in IoT Devices

TPMs can be realized as separate security chips soldered on the board or embedded in SoCs. Discrete TPM chips are common in embedded applications. For example, Microchip's FIPS- certified TPM 1.2 (the AT97SCx family) integrates an AVR microcontroller with EEPROM and security logic, and is offered in SPI, LPC or I²C packages [6]. Infineon's OPTIGA™ TPM 2.0 (e.g. SLB 9670) similarly connects via SPI and is explicitly designed for IoT/automotive use.

In practice, a host microcontroller (ARM, x86, etc.) communicates with the TPM over SPI (or I²C/LPC on legacy platforms) and loads a software stack to issue TCG commands.

5.2 Key Management Features

TPMs provide a full suite of key management capabilities critical for IoT security: Secure Key Generation and Storage: The TPM includes a hardware random number generator (RNG) and can generate any needed key (RSA, ECC, HMAC, AES, etc.) internally. All private key portions are kept within the TPM's protected memory. For example, Microsoft notes that the TPM has a "Storage Root Key" inside the chip, and the private part of the endorsement key (EK) is "never exposed to any other component, software, or user". Similarly, AWS documentation emphasizes that all secret keys (e.g. the attestation keys and storage keys) are stored in the TPM's secure enclave. By design, even if an attacker has full software control, they cannot extract or view these private keys. The TPM can also "seal" keys or data: a key can be tied to specific platform measurements, so it can only be unsealed (decrypted) by that same TPM on the same boot state. This ensures keys cannot be used if firmware has been tampered with. Key Provisioning and Lifecycle: TPMs use a hierarchy of keys. Each TPM ships with a unique Endorsement Key (EK) pair (and usually a manufacturer-issued certificate of the EK). During device manufacturing or commissioning, the OEM will "take ownership" of the TPM by setting an owner password and generating a new Storage Root Key (SRK) for that owner.

The SRK serves as a master key for encrypting subordinate keys in the TPM's hierarchy. TPM keys can be marked as migratable or non-migratable: non-migratable keys cannot ever be exported off the TPM ,ensuring long-term protection. Owners can re-provision TPMs (for example, issuing a new SRK when a device changes owners) so that previous credentials are invalidated. TPM 2.0 also supports dynamic key creation under policy constraints and can store multiple key pairs (RSA or ECC) for different uses over the device lifetime.

Identity Binding and Authentication: A core TPM capability is binding a unique identity to the device. The EK (and the TPM's manufacturer-issued certificate, if any) serve as a hardware root- of-trust. Devices can prove ownership of the EK by signing challenges, enabling a remote party to authenticate the device. For example, Azure's Device Provisioning Service uses the TPM's EK to establish device identity during enrolment. Above the EK, TPM 2.0 can generate Attestation Keys (AK) or use Attestation Identity Keys (AIKs) to create device-specific certificates. In practice, the TPM can internally generate an X.509 key pair and have the TPM's SRK protect the private key; the public key is then certified by a CA as the device's DevID. Infineon explains that the TPM safeguards private keys associated with custom X.509 certificates, which serve as Secure Device Identifiers (DevIDs) in accordance with the IEEE 802.1AR standard. In summary, the TPM hardware ensures that cryptographic identity (certificates/keys) is tied to the physical device and cannot be transferred or spoofed.

6 Enhancing IoT Key Management with BLAKE3 and TPM

Practical Integration: In a typical IoT device, a hardware TPM (Trusted Platform Module) or TPM-like secure element provides a root of trust by generating, storing and protecting cryptographic keys in silicon [4]. During provisioning, the manufacturer or OEM injects a device- specific key (for example, an RSA or ECC key) into the TPM. The TPM can then seal this root key to specific firmware measurements (PCRs) so that it can be unsealed only when the device boots with known, untampered

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

code.

In software, BLAKE3 is used as a fast, secure key derivation function: for example, a single TPM-protected master secret can be fed into BLAKE3's key-derivation mode with application-specific context strings to produce multiple session or data-encryption keys. A concrete flow might be:
1. On boot, the secure bootloader unseals the device's master secret from the TPM (ensuring PCR values match expected firmware);

2. The bootloader or OS uses BLAKE3.derive_key (context, master_secret) to derive a network-encryption key and a signing key for that boot session;

3. The device uses those keys for TLS or MQTT sessions and then discards them when done. Because BLAKE3 can output arbitrary-length keys and even act as an extensible-output function, a single derive call can produce all needed keys in one shot.

Meanwhile, the TPM can also store other sensitive values (like root CA certificates or code signing public keys) in its NV memory or secure storage. For firmware updates, the device's bootloader might verify a BLAKE3 hash of the update image against a signature: typically the manufacturer signs the firmware hash, and the public key used for verification is held inside the TPM or a secure element. The TPM can perform the signature check (if it supports the algorithm) or simply store the public key so that even if the OS is compromised, firmware can only be installed if it's properly signed.

Example Key-derivation Flow: A TPM is provisioned with a 256-bit device secret. At runtime, the OS or a trusted service reads that secret (via TPM2_Unseal after verifying PCRs). It then runs BLAKE3 in derive-key mode: K = BLAKE3.derive_key("IoT-App-Session", device_secret). This expands the secret into a new key K. That key can be used to generate session keys (e.g. by BLAKE3's XOF) or be used directly for symmetric encryption/HMAC with minimal CPU cost. If multiple keys are needed, BLAKE3's XOF can stretch K into as many bytes as required. The master device_secret remains protected by the TPM and is not exposed in RAM except transiently, and any attempt to boot with tampered firmware (PCR mismatch) would prevent TPM2_Unseal.

Sealing and Attestation: The TPM's sealing mechanism ties stored data (keys or secrets) to the device state. For instance, an IoT device might seal a symmetric key so that it will only be unsealed if the bootloader and kernel haven't changed. The TPM also supports attestation: the device can compute a cryptographic quote (TPM2_Quote) over its PCR registers and a nonce, signing them with an internal attestation key. This lets a cloud service verify the device's firmware stack remotely. In Microsoft's Azure IoT DPS, for example, the TPM's built-in Endorsement Key (EK) is used for this asymmetric authentication: the TPM has a unique EK burned into it, and DPS uses that to certify device identity. The TPM can also generate a one-time Attestation Identity Key (AIK) whose certificate is tied to the EK. In short, the TPM provides hardware-backed attestation so that a verifier can be sure the device running BLAKE3-based crypto is genuine and unmodified.

Theoretical Architecture: BLAKE3 is a modern hash function with a built-in key-derivation mode that acts as a pseudorandom function (PRF). In key-derivation mode it takes a context string and key material and produces arbitrary-length output. The context string is typically an application- unique label (e.g. "SessionKey" or "FirmwareHash") that separates different uses of the same key material. Unlike HKDF (which explicitly takes a salt and has extract-and-expand phases), BLAKE3's derive mode integrates context via two flag bits and a one-pass tree hash design. The IETF draft for BLAKE3 explicitly notes that "BLAKE3 in key derivation mode can replace HKDF".

In practice, an IoT firmware could use BLAKE3 to derive multiple subkeys from a hardware root key without the overhead of two hashing rounds per block (as in HMAC-SHA256). BLAKE3 also supports a keyed hash mode (like HMAC) if a simpler PRF is needed, and an extendable-output

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

mode for XOF applications. The TPM acts as a hardware Root of Trust. It contains a Storage Root Key (SRK) and an Endorsement Key (EK) whose private halves are never exposed outside the chip. Symmetric or asymmetric keys generated by the TPM can be marked non-migratable, meaning their private part can never leave the TPM. In addition, the TPM includes a true random number generator and health checks: for example, it injects hardware entropy into its operations to frustrate brute-force attacks.

The TPM also measures boot components: when the CPU boots, the UEFI or bootloader hashes each firmware stage (it could use SHA-256 or even BLAKE3 if supported by the software) and extends those measurements into Platform Configuration Registers (PCRs). These PCR values form a tamper-proof log of what code was loaded. Because the TPM stores measurements internally and can sign them, an external verifier can use TPM-based measured boot with attestation to ensure the device started in a known good state.

In summary, BLAKE3 provides the cryptographic computations (hashing, KDF) in software, while the TPM provides secure key storage, measurement and attestation in hardware. Together they allow an IoT device to bootstrap trust: the TPM ensures the master secrets and identity keys are protected, and BLAKE3 uses those secrets to derive all other keys needed (for encryption, MACs, etc.) in an efficient, domain-separated way. Performance and Security Analysis: BLAKE3's design delivers very high throughput with low latency. The authors report it being roughly 5× faster than BLAKE2 on single-threaded loads and over 20× faster when parallelized on large messages. Comparative benchmarks show BLAKE3 exceeding 1 GB/s per core on modern CPUs, roughly three times the speed of SHA-256 even when the latter uses hardware acceleration.

In practice, on a single-core MCU without SIMD, the speedup may be lower, but BLAKE3 is still substantially faster than HMAC-SHA256 in typical cases. Because IoT devices often run on 32- bit ARM cores, BLAKE3's 32-bit-friendly implementation (even without SIMD) can save energy and CPU cycles. HKDF-SHA256 requires two full hash computations (extract and expand), whereas BLAKE3's derive mode is a single pass, further reducing CPU and memory overhead. In key-derivation benchmarks (e.g. JS libraries), BLAKE3-based KDFs have been shown to run hundreds of thousands of operations per second, much higher than HKDF-SHA256 implementations.

Importantly, BLAKE3 has no known weaknesses: it builds on the well-analyzed BLAKE2/Skein family and passes all practical cryptanalysis to date. The IETF draft explicitly states that BLAKE3 is "designed to be as secure as BLAKE2" while gaining performance via fewer rounds and a Merkle tree construction. The TPM, by contrast, does add some latency for crypto operations (e.g. RSA signing may be slower than in pure software), but it provides security guarantees that a software-only key cannot. The private keys in a TPM are non-exportable and isolated from the OS. Even if malware compromises the OS, it cannot read the TPM's sealed secrets or use its keys without authorization. TPM chips incorporate physical anti-tampering features: they have dedicated enclosures, power analysis resistance, and internal sensors so that fault injection or probing will wipe keys. They also have anti-hammering logic: if an attacker tries repeatedly to guess a PIN or authorization value, the TPM will lock out further attempts. In effect, the TPM ensures key integrity and resilience: platform secrets are protected against local and side-channel attacks. Many TPMs meet FIPS or Common Criteria standards, giving confidence for critical deployments. Comparative Insight: In summary, BLAKE3 accelerates and simplifies key derivation in IoT firmware compared to legacy KDFs. Unlike plain SHA-256 or HMAC, BLAKE3's single-pass, SIMD-optimized design makes it far more efficient in CPU and energy.

In battery- or performance-constrained devices, this means faster boot-up and quicker establishment of secure channels. Meanwhile, the TPM anchors the device's identity and keys in hardware. Keys stored in the TPM are (by design) beyond the reach of the normal attack surface – a distinct advantage over purely software key stores. The TPM's measured-boot and attestation features complement

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

BLAKE3's cryptographic functions by ensuring that the keys being derived are only released when the system is in a secure state. For IoT security, this combination means that every derived key or signature has a hardware-backed root: firmware images can be verified or rolled back using TPM PCRs, and session keys can be renewed or revoked knowing the device is genuine. As Arrow Electronics notes, modern IoT edge systems use TPMs for secure boot, OTA updates, data encryption, and device authentication, precisely addressing threats like firmware manipulation or counterfeit devices.

In practice, deploying BLAKE3 for KDFs (to replace HKDF/SHA256) together with a TPM (for root-of-trust and secure key storage) yields a scheme where key material is generated efficiently in software but never fully exposed, while hardware protects the long-term secrets and integrity of the platform.

7 Conclusion

In this paper, we presented a hybrid key-management architecture that leverages the Trusted Platform Module (TPM) as a hardware root of trust and the BLAKE3 hash function for lightweight, secure key derivation in resource-constrained IoT devices. By integrating TPM-based attestation with a BLAKE3-driven HKDF, our scheme ensures that private key material remains securely confined within the hardware, while enabling fast and memory-efficient session key generation. Our experimental results on an ARM-based microcontroller demonstrate that the proposed solution introduces minimal latency and energy overhead, while providing robust protection against replay, man-in-the-middle, and impersonation attacks. With less than 15 ms of attestation overhead and sub-millisecond key derivation performance using only 12 kB of RAM, the scheme proves both practical and scalable for next-generation IoT environments. These findings validate that strong, forward-secure, and hardware-anchored key management is not only achievable but highly efficient in constrained settings, paving the way for secure and trustworthy IoT deployments at scale.

References

[1] Enhancing IoT Security: An Innovative Key Management System for Lightweight Block Ciphers MuhammadRana*,QuaziMamun and RafiqulIslam

[2] BLAKE3 one function, fast everywhere, Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn

[3] Trusted Platform Module A Survey, Kenneth Ezirim, Wai Khoo, George Koumantaris, Raymond Law, and Irippuge Milinda Perera

[4] Analyzing Key Management Solutions for IoT Environments S. Vaishnavi1, E. Angel Mary2,

[5] M. Boomalini3, Dr.Amala Dhaya M.D4 1,2,3Students, Information Technology, Loyola Institute of Technology and Science, Kanyakumari, Tamil Nadu 4Assistant Professor, Information Technology, Loyola Institute of Technology and Science, Kanyakumari, Tamil Nadu.

[6] A Review on Key Management and Lightweight Cryptography for IoT Mohammad Ayub Latif; Maaz Bin Ahmad; Muhammad Khalid Khan 10.1109/GCWOT49901.2020.9391613.

[7] Security in Internet of Things: Issues, Challenges, and Solutions July 2019Advances in Intelligent Systems and Computing Hanan Aldowah University of Science Malaysia Shafiq Ul Rehman Kingdom University Irfan Umar DOI:10.1007/978-3-319-99007-1_38.

[8] Interoperability in IOT: challenges, strategies and future direction Deep Manish Kumar Dave Independent Researcher, Bharath Kumar Mittapally Independent Researcher.

[9] An Efficient Key Management Technique for the Internet of Things Tamanna Tabassum 1, S K

Journal of Information Systems Engineering and Management
2024, 9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Alamgir Hossain 1, Md Anisur Rahman 1, Mohammed F Alhamid 2 3, M Anwar Hossain 3

[10]DOI: 10.3390/s20072049. An Effiecient Key Management Technique

[11] The Cryptographic Key Distribution System for IoT Systems in the MQTT Environment, Janusz Furtak, DOI: 10.3390/s23115102.