A Comprehensive Framework for Building Highly Secure, Network-Connected Devices: Chip to App

Khan Reaz and Gerhard Wunder

Dept. of Mathematics and Computer Science, Freie Universität Berlin, Germany.

Abstract

The rapid expansion of connected devices has amplified the need for robust and scalable security frameworks. This paper proposes a holistic approach to securing network-connected devices, covering essential layers: hardware, firmware, communication, and application. At the hardware level, we focus on secure key management, reliable random number generation, and protecting critical assets. Firmware security is addressed through mechanisms like cryptographic integrity validation and secure boot processes. For secure communication, we emphasize TLS 1.3 and optimized cipher suites tailored for both standard and resource-constrained devices. To overcome the challenges of IoT, compact digital certificates, such as CBOR, are recommended to reduce overhead and enhance performance. Additionally, the paper explores forward-looking solutions, including post-quantum cryptography, to future-proof systems against emerging threats. This framework provides actionable guidelines for manufacturers and system administrators to build secure devices that maintain confidentiality, integrity, and availability throughout their lifecycle.

Keywords: Network Security, Secure IoT, Cryptographic Framework, Post-Quantum Solutions

1 Introduction

The growing prevalence of network-connected devices in diverse industries such as healthcare, manufacturing, and smart infrastructure has significantly increased security concerns. Devices forming the Internet of Things (IoT) ecosystem are often resource-constrained, making them vulnerable to a wide range of attacks, including man-in-the-middle (MitM) attacks, ransomware, and side-channel exploits. To ensure

the safety and reliability of these systems, a holistic approach to network security is essential.

This paper introduces a comprehensive security framework that addresses four critical layers of network-connected devices:

Hardware: Ensuring secure key management, random number generation, and hardware-based root-of-trust mechanisms.
Firmware: Enforcing cryptographic integrity validation, secure boot processes, and protection against tampering.
Communication: Implementing Transport Layer Security and compact certificates optimized for resource-constrained environments.
Application: Achieving end-to-end encryption and message integrity validation while maintaining performance efficiency.

In addition to current security standards, we explore emerging post-quantum cryptographic solutions to future-proof connected devices against potential quantum computing attacks. These recommendations are designed to provide a practical, scalable, and forward-looking approach to securing the next generation of network-connected systems. The remainder of the paper is organized as follows: Section 2 defines key terminologies and requirements for secure network design. Section 3 outlines essential cryptographic concepts and attack scenarios. Section 4 provides actionable recommendations for secure implementation across hardware, firmware, and network layers. Section 5 compares current cryptographic standards with post-quantum cryptographic advancements. Finally, Section 6 concludes the paper and highlights future research directions.

2 Terminologies and Requirements

This section introduces the foundational terms and requirements essential for designing and implementing secure network-connected devices.

2.1 Components

User

The user, or often known as end-user, is a human entity who is legally registered to some authority by proving their identity.

Device

A device refers to any physical entity equipped with computational capabilities that can perform data processing, communication, and interaction within a network. Devices encompass a broad range of hardware, from traditional computing systems like desktops and servers to mobile phones, IoT sensors, and embedded systems.

Authenticator

An authenticator is a handheld smart device, such as a smartphone or tablet, equipped with a rich user interface and capable of communication through multiple radio interfaces, including Wi-Fi, cellular networks, and Bluetooth.

Digital ID/Smart eID

A digital ID, or smart eID, functions as the digital counterpart of a physical ID card, stored on a smart device.

eID Service

The eID service fulfills several critical functions, including managing communication with the ID card's chip, retrieving and updating authorization certificates and revocation lists, and securely transmitting verified data.

eID Service Provider

Service providers, such as hospitals, hotels, and public transportation companies, can operate their own eID servers by obtaining accreditation from the eID issuer.

eID Server

The eID server comprises both hardware and software components operated by the service provider to integrate eID functionality into its IT infrastructure. Its primary functions include establishing secure communication with client software and the ID card's chip, transmitting data to the specified service, verifying the ID card's validity and authenticity, checking for any blocks imposed by the ID card holder, and communicating the eID function results to the provider's other systems.

Additionally, the eID server regularly receives updated authorization certificates and revocation lists from the certification provider, ensuring ongoing compliance and security. Service providers wishing to develop their own eID servers must ensure that both hardware and software components comply with the Technical Guidelines established by the Federal Office for Information Security. This compliance is crucial for executing cryptographic protocols with the ID card's chip and for routinely receiving necessary authorization certificates and revocation lists.

Chip Authentication

The Chip Authentication process establishes a secure connection between RF chips and readers and enables detection of cloned RF chips within identification documents. Each RF chip supporting this method possesses a unique public-private key pair, with the private key securely stored in a protected area, inaccessible even in the event of cloning.

During the Chip Authentication process, the RF chip transmits its public key along with a random number to the reader, which subsequently generates its own key pair. By using their respective private keys, the exchanged public key, and the random number, both the RF chip and reader compute a shared secret key. This shared key provides robust encryption for all subsequent communication, thereby ensuring data security. The reader then authenticates the chip's private key through this shared secret. If a cloned RF chip attempts to generate a new set of keys, Passive Authentication can detect this alteration, as the public key is protected from unauthorized modification by a digital signature.

Passive Authentication

Passive Authentication (PA) is a process used to verify the authenticity and integrity of the data stored on the eID chip.

When an eID is issued, its data is digitally signed using a Document Signer certificate, provided by the Country Signing Certificate Authority (CSCA) of the issuing country. Access to this certificate is restricted to entities authorized by the issuing nation, ensuring that only validated data is signed and stored on the eID. The CSCA certificate acts as the root of the country's public key infrastructure (PKI), establishing a certificate hierarchy that certifies the legitimacy and accuracy of data on official identification documents.

During the eID verification process, Passive Authentication confirms the digital signature on the chip's data, linking it back to the CSCA certificate. This procedure verifies that the information on the RF chip is both authentic and securely issued by an authorized document producer.

Server/Back-end

An IoT analytics provider may operate its services within its own cloud infrastructure or within a hosted cloud environment. The physical or virtual instance of this service is referred to as the server (S). This server is accessible via the internet, or, in the case of a local installation, it can be discovered within the local network. Access to the server's APIs is secured through authentication and authorization mechanisms, ensuring secure communication and reliable data handling.

Enrollee

An Enrollee is a newly manufactured device that has not yet been provisioned. It is equipped with IP-based network connectivity and includes at least one wireless interface, such as Wi-Fi, Ultra-Wideband, or Ethernet. This interface supports secure communication during the initial setup and onboarding process.

Gateway/Router

A Gateway or Router manages data traffic within the local network and provides a connection bridge to external networks. Accessible through IP-based connections, it serves as both a routing and network access point.

Beyond routing data, the gateway enforces security measures such as authentication, firewalling, and access control, ensuring that network traffic is both efficiently managed and secure.

System Administrator

A System Administrator (SysAdmin) is responsible for managing and maintaining an organization's core IT infrastructure, ensuring the security, reliability, and optimal performance of servers, networks, and connected devices. The SysAdmin oversees tasks such as configuring and deploying servers, managing user access, maintaining software updates, and implementing security protocols.

In the context of cryptographic operations, the SysAdmin plays a vital role in initiating and managing key processes, including generating cryptographic key pairs, creating and submitting Certificate Signing Requests (CSRs), and installing certificates on the appropriate servers.

3 Definitions

This section introduces fundamental cryptographic concepts and highlights critical attack types that secure network architectures must withstand.

Universally unique identifier (UUID) is a 128-bit number used to identify information in computer systems. The term globally unique identifier (GUID) is also used.

A **root of trust (RoT)** is a set of functions that are always trusted by an operating system. It serves as the foundation for all secure operations in a computing system. A RoT contains keys used for digital signing and verification, as well as cryptographic functions that enable secure boot processes. It is an essential security asset. Embedding an RoT in hardware provides a trusted execution environment and creates a solid foundation for electronic systems security.

Key is a piece of information (for example, a randomized bit-string) required to encrypt or decrypt data.

Key management system (KMS) is used to manage cryptographic keys, including their generation, storage, use, rotation, destruction, and replacement.

Key type is a specific implementation type for a cryptographic primitive.

Plaintext: Any infromation in its original form.

Primitive is cryptographic building block that manages an underlying algorithm so users can perform cryptographic tasks safely.

Symmetric key encryption: A cryptographic algorithm that uses the same key to encrypt plaintext and decrypt ciphertext.

Asymmetric key encryption: A cryptographic system that uses paired keys—public and private—to encrypt and decrypt data. Public keys are used to encrypt data and may be shared. Private keys are used to decrypt data, and are only known to the owner.

Ciphertext: The result of encryption performed on plaintext using an algorithm. Ciphertext is not understandable until it has been converted back into plaintext using a key.

Deterministic encryption: A type of encryption that always produces the same ciphertext for a given plaintext and key. This can be risky, because an attacker only needs to find out which ciphertext corresponds to a given plaintext input to identify it.

Hybrid encryption: A cryptographic system that combines asymmetric key encryption and symmetric key encryption. Hybrid encryption combines the efficiency of symmetric encryption with the convenience of public-key encryption. To encrypt a message, a fresh symmetric key is generated and used to encrypt the plaintext data, while the recipient's public key is used to encrypt the symmetric key only. The final ciphertext consists of the symmetric ciphertext and the encrypted symmetric key.

Adaptive Chosen Ciphertext Attack: An adversary with access to a decryption functions attempts to defeat the security of a scheme to which the function belongs. The attacker has the capability to continually access the oracle and get a response for polynomially many arbitrary ciphertext.

Perfect forward secrecy is commonly used to denote a feature of key agreement protocols which gives assurances that past session keys will not be compromised even if the private key is compromised.

Digital Signature confirms the authenticity and the integrity of the data by signing it with key. It uses asymmetric keys: private key is used for signing and public key is used for verifying.

Zero-Trust Model or Zero-Trust architecture is a collection of concepts designed to minimise uncertainty in enforcing accurate access decisions in compromised networks.

Phishing attacks: These attacks involve an attacker sending a fake email or message that appears to be from a legitimate source, such as a bank or social media platform, with the intention of tricking the recipient into sharing sensitive information, such as passwords or financial details.

Malware attacks: Malware is a type of software designed to damage or gain unauthorized access to a computer system. Examples of malware include viruses, Trojan horses, and ransomware.

Denial of Service (DoS) attacks: These attacks involve flooding a network or website with traffic, making it unavailable to users. Distributed Denial of Service (DDoS) attacks involve multiple sources flooding the targeted network or website.

Man-in-the-middle (MitM) attacks: These attacks involve an attacker intercepting communications between two parties and potentially altering the information being exchanged.

Password attacks: These attacks involve attempting to guess or crack passwords to gain unauthorized access to a system or account.

SQL injection attacks: These attacks involve exploiting vulnerabilities in a website or application's code to gain unauthorized access to its underlying database.

Cross-site scripting (XSS) attacks: insert malicious code into a legitimate website or application script to get a user's information, often using third-party web resources. Attackers frequently use JavaScript for XSS attacks, but Microsoft VCScript, ActiveX and Adobe Flash can be used, too.

DNS tunneling is used by cybercriminals to exchange application data, like extract data silently or establish a communication channel with an unknown server.

Backdoor Trojan creates a backdoor vulnerability in the victim's system, allowing the attacker to gain remote, and almost total, control. Frequently used to link up a group of victims' computers into a botnet or zombie network, attackers can use the Trojan for other cybercrimes.

Ransomware is sophisticated malware that takes advantage of system weaknesses, using strong encryption to hold data or system functionality hostage. Cybercriminals use ransomware to demand payment in exchange for releasing the system. A recent development with ransomware is the add-on of extortion tactic.

Zero-day exploit attacks take advantage of unknown hardware and software weaknesses. These vulnerabilities can exist for days, months or years before developers learn about the flaws.

p-value is a statistical measure that quantifies the probability of observing a test value that is at least as extreme as the particular value that has just been observed (tail probability) if the null hypothesis is true. If this p-value is smaller than a predefined bound, it indicates that the null hypothesis should be rejected.

A **botnet** is defined as a group of internet-connected devices that have been subjected to a hacking attack and are subsequently controlled as a collective without the knowledge or consent of the device owner. These devices are typically used for the purpose of carrying out malicious activities.

Confidentiality ensures that information is only accessible to authorized parties and is kept secret from unauthorized entities.
Example: Encryption ensures that only intended recipient(s) can read the content.

Integrity guarantees that the data has not been altered or tampered with during transmission or storage.
Example: Message Authentication Codes (MACs), digital signatures, and hash functions are commonly used to verify integrity.

Authentication checks that the parties involved in communication are who they claim to be.
Examples: Passwords, biometrics, and digital certificates

Authorization determines whether an authenticated entity has permission to perform a specific action or access specific resources.
Examples: Role-based access control (RBAC) and Access control lists (ACLs).

Non-repudiation makes sure that a party cannot deny the authenticity of their signature on a document or a message that they originated.
Examples: Digital signatures, logging mechanisms.

Availability means that systems and data are available to authorized users when needed, preventing disruptions from attacks or failures.
Examples: Redundancy and failover systems, protection against Denial of Service (DoS) attacks.

Forward Secrecy dictates that session keys will not be compromised even if long-term keys are compromised in the future.
Examples: Advanced Diffie-Hellman key exchange in protocols like TLS.

Anonymity means that the identity of a user remains hidden from other users or third parties.
Examples: Anonymity networks like Tor, anonymous credentials.

Unlinkability occurs when an adversary cannot link two or more actions, messages, or identities to a single entity.
Examples: Mix networks, onion routing.

Unobservability ensures that an adversary cannot detect whether a particular action or communication is taking place.
Examples: Masking channels.

Accountability ensures that actions of individuals or systems can be traced to them, and they can be held responsible for their actions.
Examples: Logging and auditing systems, cryptographic signatures.

Verifiability ensures that the correctness of operations or transactions can be independently verified by any party.
Examples: Verifiable voting systems, publicly verifiable encryption schemes.

Auditability ensures that actions and events can be recorded and reviewed, typically for security, compliance, or forensic purposes.
Examples: System logs, blockchain technology.

Trustworthiness ensures that the system or component can be relied upon to perform its intended function securely and correctly.
Examples: Trusted platform module (TPM), formal verification of protocols.

Robustness ensures that the system can withstand attacks, failures, or environmental changes without compromising security.
Examples: Fault-tolerant systems, resilience against side-channel attacks.

Resistance to Side-Channel Attacks means that the system is resistant to attacks that exploit unintended information leakage, such as timing, power consumption, or electromagnetic emanations.
Examples: Constant-time algorithms, shielding and noise injection.

Resistance to Replay Attacks means that an attacker cannot reuse a previously intercepted message to deceive the system or its participants.
Examples: Nonces, timestamps, sequence numbers.

Resistance to Impersonation Attacks ensures that an attacker cannot successfully masquerade as another user or entity.
Examples: Multi-factor authentication, Public key infrastructure (PKI).

Resistance to Collusion ensures that even if multiple adversaries or compromised entities collude, they cannot break the security of the protocol.
Examples: Secret sharing schemes, distributed consensus protocols.

Resilience to Denial-of-Service (DoS) Attacks ensures that the system remains operational or degrades gracefully under attack conditions.
Examples: Rate limiting, CAPTCHA, traffic analysis.

Ephemeral Key Security ensures that the security of a session is not compromised if the ephemeral (temporary) keys are exposed or lost.

Resistance to Cryptanalysis ensures that cryptographic schemes are resistant to attacks that aim to break the encryption, such as brute-force attacks, differential cryptanalysis, or algebraic attacks.
Examples: Use of strong cryptographic algorithms ChaCha20, continual updates to cryptographic standards.

Resistance to Traffic Analysis means that an adversary cannot gain useful information by analyzing patterns in the flow of communications.
Examples: Padding, randomization techniques, onion routing.

Formal Verification and Proof of Security ensures that the security properties of a protocol or system can be mathematically proven, usually within a formal model or framework.
Examples: Formal methods, proof assistants like ProVerif or Tamarin.

Indistinguishability under Chosen Plaintext Attack (IND-CPA) formalizes the security of an encryption scheme against an adversary who can choose plaintexts to be encrypted. The scheme is considered secure if the adversary cannot distinguish between the encryptions of two chosen plaintexts, even when they have access to an encryption oracle that provides encryptions of plaintexts of their choice. Essentially, even if the adversary can see the ciphertexts of any plaintexts they select, they should not be able to gain any advantage in distinguishing between the encryptions of two specific plaintexts.

Indistinguishability under Chosen Ciphertext Attack (IND-CCA) formalizes the security of an encryption scheme against an adversary who can adaptively choose ciphertexts to be decrypted, except for the challenge ciphertext. The scheme is considered secure if the adversary cannot distinguish between the encryptions of two chosen plaintexts, even with access to a decryption oracle.

Indistinguishability under Adaptive Chosen Ciphertext Attack (IND-CCA2) provides a formal framework for ensuring the security of an encryption scheme against an adversary who can adaptively choose ciphertexts to be decrypted, both before and after receiving a challenge ciphertext, except for the challenge ciphertext itself. The scheme is deemed secure if the adversary is unable to distinguish between the encryptions of two selected plaintexts, even with adaptive access to both encryption and decryption oracles. This implies that, despite having the capability to decrypt any ciphertexts (except the challenge), the adversary cannot confirm any information

that would assist them in differentiating between the encryptions of the two chosen plaintexts.

4 Recommendations on procedures

In addition to presenting security protocols, this work aims to offer a set of clear guidelines and technical requirements to establish a balanced level of cryptographic security applicable to all network-connected devices manufactured by device producers, regardless of industry. We have identified ***five*** essential characteristics for achieving highly secure, network-connected devices:

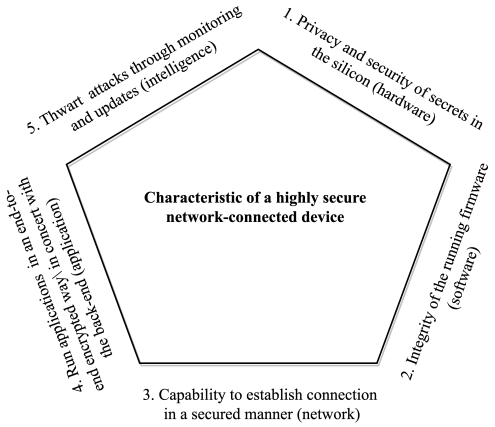

Fig. 1 Five characteristics of a secure-networked device

1. *Privacy and security of secrets in the silicon* ***(hardware)***
2. *Integrity of the running firmware* ***(software)***
3. *Capability to establish connection in a secured manner* ***(network)***
4. *Run* ***applications*** *in an end-to-end encrypted way*
5. *Thwart attacks through* ***intelligent*** *monitoring and updates*

In this work, we concentrate on the first four key areas: hardware, firmware, network, and applications. The fifth area, intelligence, is reserved for future research.

4.1 Random Number Generation

In industry, two primary standards guide the design and evaluation of both deterministic and non-deterministic random number generators (RNGs): the guidelines provided by the National Institute of Standards and Technology (NIST) [1] and the AIS 20/31 specification issued by the German Federal Office for Information Security (BSI)[2]. It is recommended that readers consult these specifications to verify compliance of the RNG implementation on a given device.

4.2 Cryptographic assets generation and storage

The generation and injection of a device root key into a SoC typically occur during the manufacturing process. This procedure must include the following essential steps:

- A RNG creates a unique private key for each device. The RNG relies on a high-entropy source, such as a hardware-based true random number generator (TRNG), to ensure that the key is random and unique.
- A cryptographic algorithm, such as ECC, is used to generate a corresponding public key from the private key.
- The public key is then signed by a trusted authority, such as a CA, resulting in a digital certificate that verifies the authenticity of the device.
- The signed digital certificate is injected into the device during manufacturing and is typically stored in non-volatile memory, such as One-Time Programmable (OTP) memory.
- The private key is securely stored within a secure element, such as a TPM or a HSM, integrated within the SoC, ensuring protection against unauthorized access.

The device root key is used for various security functions, including secure boot, device authentication, and secure communication. It also serves as the basis for deriving additional keys, such as session keys and content encryption keys, which are used to secure communication and data. The secure element storing the key is designed to prevent extraction or tampering and allows access only through secure channels.

In this setup, TPM technology or secure storage hardware provides a hardware-based layer of security. For instance, a TPM chip includes a dedicated cryptographic processor that safeguards cryptographic keys and digital certificates, offering strong protection against unauthorized access and key compromise.

4.3 Generation of Private-Public Keys

In 2019, Keyfactor conducted a study analyzing 75 million digital certificates using RSA keys, finding that 1 in every 172 certificates was vulnerable to attacks capable of exposing the private key [3]. This vulnerability was primarily due to weak random number generation. A significant portion of these compromised certificates was identified in IoT and embedded devices, such as firewalls, routers, and switches. Insufficient randomness in the prime numbers used to generate RSA public keys can result in two distinct keys sharing a common factor, which enables attackers to easily derive the remaining factors and thereby compromise the keys.

Public-private key pairs can serve two main purposes: generating and verifying digital signatures, or establishing secure keys for transporting symmetric keys. Each key pair is uniquely associated with an entity, referred to as the key-pair owner. Key pairs can be generated through one of the following methods:

- The key-pair owner generates the keys independently in a secure environment, ensuring the secrecy of the private key.

– Alternatively, a third party, such as an identity provider, may generate the key pair alongside the owner. When a key pair is intended for use by an individual, it should be protected by a *password* known only to the owner, minimizing the risk of unauthorized use through copying.

4.4 Appropriate certificate issuance process

An organized and secure certificate issuance process is essential for establishing a trusted TLS environment. This section outlines the recommended steps to ensure that the certificate generated for a server is authenticated, verified, and securely deployed. Each step, from the initial generation of cryptographic key pairs to the installation of the certificate on the server, is designed to maintain the integrity of the server's identity and the security of its communications.

– The system administrator (*SysAdmin*) initiates the cryptographic key pair generation process on the TLS server using designated server utilities, thereby creating a public key and a private key by choosing the correct algorithm according to the provided guideline.
– The *SysAdmin* inputs the required domain name into the utilities. This action generates a CSR that encapsulates the server's address and its public key. Subsequently, the SA extracts the CSR from the server and store as a file.
– The CSR is then submitted to the legal Registration Authority (RA), who is responsible to review and approve the certificate request.
– Upon receiving the CSR, the RA conducts a thorough evaluation, verifying the request's legitimacy and the requester's authorization. Following successful validation, the RA conveys an approval to the Certificate Authority (CA).
– The CA, in response to the approval, proceeds to issue the certificate.
– The SA is notified by the CA of the certificate's availability, either through an email containing the certificate or via a link for its download. The administrator then acquires the server certificate. Additionally, the SA also obtains the CA certificate chain from the CA.
– The acquired server certificate and the certificate chain is then installed on the newly configured server by the SA.

Remarks on Certificate

– As per the NIST guideline [4], after 1 January, 2024, only TLS 1.3 shall be used to ensure highest security in transport layer.
– Certificates should be signed with same signature algorithms.
– The validity period for end-entity certificates should not exceed 398 days (approximately 13 months) to enhance security.

4.5 Compact digital certificates for IoT

Implementing Public Key Infrastructure (PKI) in Internet of Things (IoT) environments presents challenges due to the size and encoding of standard X.509 public key certificates, which are often too large for constrained devices and networks [5]. To

address this, adopting more compact certificate formats, such as the CBOR, offers a solution. CBOR encoding can significantly reduce certificate size, leading to performance benefits such as reduced communication overhead, lower power consumption, decreased latency, and optimized storage—features essential for resource-constrained IoT systems [6]. Maintaining compatibility with X.509 certificates during a transitional phase may also be beneficial, given the standard's widespread use in PKI. By employing efficient encoding formats, IoT implementations can achieve enhanced security and reliability without exceeding device and network limitations.

4.6 Integrity and Authentication Protection

Cipher-based Message Authentication Code (CMAC) and Hash-based Message Authentication Code (HMAC) are widely used to provide integrity and authenticity to messages. While both serve similar purposes, they differ in underlying construction, key requirements, security properties, and performance. CMAC can be more efficient in scenarios where block cipher acceleration (such as AES) is available, whereas HMAC offers flexibility and is widely supported across platforms, often providing a higher level of security depending on the application requirements.

CMAC relies on a symmetric encryption algorithm, such as AES, to generate a fixed-length tag (typically 128 bits with AES) that is appended to the message. This tag is computed using a shared secret key with the block cipher in a specific mode. The recipient can then verify the integrity of the message by recalculating the tag with the same key. One advantage of CMAC is that it uses the same key as the underlying cipher, simplifying key management.

In contrast, HMAC is based on cryptographic hash functions, such as SHA-2 or SHA-3. A separate secret key is used with the hash function to compute a tag whose length depends on the hash function used (e.g., 256 bits for SHA-256). HMAC's construction ensures that even if the hash function has minor weaknesses, HMAC remains secure, making it a highly reliable choice for a range of applications.

Network Connection Establishment Procedure

Transport Layer Security (TLS) is the standard cryptographic protocol for authenticating and encrypting communications between clients and servers. TLS relies on digital certificates, which contain verifiable information about the certificate holder's identity along with an associated private key, essential for establishing secure communication channels.

For effective TLS implementation, a server must possess both an authenticated certificate and its corresponding private key. Together, these components enable the server to verify its identity to clients and are crucial for generating symmetric encryption keys. These keys allow the secure encryption and decryption of transmitted data, ensuring confidentiality and integrity in client-server communications. The overall security of the TLS framework relies on two primary factors: secure implementation

and configuration of TLS servers, and effective management of TLS certificates. Managing TLS certificates is a complex task that requires continuous monitoring, timely updates, and renewals to prevent security vulnerabilities. Inadequate maintenance practices can lead to serious risks, such as service disruptions, increased susceptibility to cyberattacks, and compromised data integrity.

When initiating a secure connection, the client and server engage in a negotiation process to select an appropriate cipher suite. The client initiates this by sending a handshake message containing a list of supported cipher suites. The server then selects one based on its own preference order and responds with a handshake message specifying the chosen suite. It's important to note that the server may not always select the strongest cipher suite proposed by the client, as it may prioritize differently. Consequently, there's no guarantee that the most secure cipher suite will be chosen. If no mutually acceptable cipher suite is found, the connection will terminate.

To comply with security guidelines, the server must meet specific requirements across several categories, including TLS protocol versions, server keys and certificates, cryptographic functionality, TLS extension support, client verification, session continuation, compression methods, and operational standards.

- All device categories (i.e., Standard and Constrained Devices) must use ephemeral keys to provide perfect forward secrecy, without exception.
- The server must perform revocation checking of client certificates whenever client certificate authentication is used.
- Constrained devices that do not support TLS 1.3 should be deployed in configurations that maintain security, even if they use alternative TLS 1.2 versions
- TLS 1.3 should not be used with the 0-RTT option unless absolutely necessary, as 0-RTT introduces potential security risks, such as replay attacks.

The following cipher suites shall be used exclusively with elliptic curve-based server certificates. In TLS 1.3 naming conventions, for instance, `TLS_AES_256_GCM_SHA384` indicates that messages are encrypted and authenticated using AES-256 in Galois/-Counter Mode (GCM), with SHA-384 applied in the HMAC-based Key Derivation Function (HKDF) process.

TLS Cipher Suites	
Standard Device	*Constrained Device*
`TLS_AES_128_GCM_SHA256`	`TLS_AES_128_GCM_SHA256`
`TLS_AES_256_GCM_SHA384`	
`TLS_CHACHA20_POLY1305_SHA256`	

Table 1 Recommended TLS Cipher Suites for Standard Devices and Constrained Devices

5 Recommended cipher suite based on operation mode

To remain adaptable within the rapidly evolving cryptographic landscape, this architecture includes two operational modes. The first, **Current Mode**, employs well-established protocols and standards that satisfy our outlined security requirements, providing both robustness and wide deployability. The second, **Future Mode**, is designed to accommodate emerging algorithms and protocols that have undergone rigorous multi-year evaluations through initiatives such as the NIST Post-Quantum Cryptography Standardization Project. While these algorithms are still being finalized by NIST and other regulatory authorities, adhering to their specifications as they become available will ensure long-term security and compliance.

5.1 Current Mode

This mode incorporates cryptographic primitives selected for their broad support across platforms and libraries, as well as extensive analysis and standardization by NIST and other authoritative bodies. These algorithms provide robust security against known attacks while remaining efficient in terms of computation and communication.

Current Mode	
Type	***Specification***
Symmetric-key encryption:	AES-128, AES-256bit GCM [7]
Public-key algorithms:	Ed25519 (128 bits), Ed448 (224 bits) [8]
Digital signatures:	EdDSA [8], [9]
Integrity & Authentication:	CMAC [10], HMAC
Key derivation functions:	As per NIST SP 800-56C [11]
Random number generation:	NIST SP 800-90A-C [1]

Table 2 Recommended Cryptographic Algorithms and Standards for *Current Mode*

5.1.1 Recommendations for *Current Mode*

Public-private key pairs should be generated using elliptic curve cryptography, specifically with Edwards-curve-25519 (Ed25519) and Edwards-curve-448 (Ed448) [12]. Ed25519 provides a security level equivalent to 128 bits, while Ed448 offers a security level of approximately 224 bits.

The EdDSA (Edwards-curve Digital Signature Algorithm) has multiple advantages, including high performance across diverse platforms and resilience to side-channel attacks. Its compact public key (32 bytes) and signature size (64 bytes) make it particularly suitable for resource-constrained devices. Edwards curves, like Ed25519 and Ed448, are generally preferred over traditional elliptic curves, such as the NIST P-256 and its variants, due to their more robust security properties and simplicity in implementation.

A significant advantage of EdDSA is its collision resilience, which means that hash-function collisions do not compromise system security. Additionally, the formulas for Edwards curves are complete, eliminating the need for EdDSA to perform computationally expensive point validation on untrusted public values, thus enhancing both efficiency and security.

5.2 Future Mode

The cryptographic primitives for **Future Mode** are selected based on their performance and security as demonstrated in NIST's Post-Quantum Cryptography Standardization Project. These include algorithms that are resistant to quantum computing attacks and are expected to serve as foundational standards in the coming years. Given the relatively recent development of these algorithms, they may undergo further changes, so continuous monitoring of updates from NIST and similar bodies is recommended.

While NIST indicates that SHA-2 family hash functions, such as SHA-256, SHA-384, and SHA-512, are likely secure beyond 2030, we recommend adopting the SHA-3 family functions for additional resilience against future threats.

Future Mode	
Type	***Specification***
Symmetric-key encryption	AES-256bit GCM
Key Encapsulation Mechanism	ML-KEM [13]
Digital signatures	ML-DSA [14], SLH-DSA [15]
Hash functions	SHA-3(384,512) [16]
Random number generation	NIST SP 800-90A-C [1]

Table 3 Recommended Cryptographic Algorithms and Standards for *Future Mode*

6 Conclusion

In this paper, we laid out a comprehensive framework for securing network-connected devices, targeting critical areas like hardware, firmware, communication, and application security. Starting with secure hardware designs, we highlighted the importance of entropy-based random number generation, secure key management, and root-of-trust implementations. At the firmware layer, techniques such as cryptographic signatures and secure boot processes ensure only validated software runs on devices. For communication, we underlined the necessity of TLS 1.3 and recommended specific cipher suites like AES-GCM and ChaCha20-Poly1305 to meet the needs of both standard and constrained devices. By advocating for compact digital certificates, like CBOR, we addressed the challenges posed by resource-limited IoT systems. Finally, we explored future-proof solutions such as post-quantum cryptographic algorithms to address long-term security concerns in a post-quantum world. Together, these recommendations equip device manufacturers and network architects with the tools needed to build secure, resilient systems that adapt to evolving threats. Moving forward,

research efforts will focus on integrating real-time monitoring, automated updates, and intelligent threat detection to strengthen the overall security posture of connected devices.

References

[1] Barker, E., Kelsey, J.: Recommendation for random number generation using deterministic random bit generators. Technical Report NIST Special Publication 800-90A Revision 1, National Institute of Standards and Technology (June 2015). Accessed: 2024-10-23. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[2] Peter, M., Schindler, W.: A Proposal for Functionality Classes for Random Number Generators. Technical guideline, Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany (2024). https://www.bsi.bund.de/dok/randomnumbergenerators

[3] Kilgallin, J., Vasko, R.: Factoring rsa keys in the iot era. In: 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), pp. 184–189 (2019). IEEE

[4] McKay, K.A., Cooper, D.A.: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. National Institute of Standards and Technology, Gaithersburg, MD, **NIST SP 800-52r2** (2019) https://doi.org/10.6028/NIST.SP.800-52r2

[5] Bormann, C., Ersue, M., Keranen, A.: RFC 7228: Terminology for constrained-node networks. RFC Editor (2014)

[6] Bormann, C., Hoffman, P.: Rfc 8949: Concise binary object representation (cbor). Internet Engineering Task Force (IETF) (2020)

[7] Dworkin, M.J.: Recommendation for block cipher modes of operation: Galois/-counter mode (gcm) and gmac (2007)

[8] Chen, L., Moody, D., Regenscheid, A., Randall, K.: Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Technical report, National Institute of Standards and Technology (2019)

[9] National Institute of Standards and Technology (NIST): Digital Signature Standard (DSS). (Department of Commerce, Washington, DC.). Federal Information Processing Standards Publication **FIPS 186-5** (2023)

[10] Dworkin, M.J.: Recommendation for block cipher modes of operation: The cmac mode for authentication (2016)

[11] Barker, E., Chen, L., Davis, R.: Recommendation for key-derivation methods

in key-establishment schemes. Technical report (2020). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf

[12] Josefsson, S., Liusvaara, I.: Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, (2017). http://www.rfc-editor.org/info/rfc8032

[13] National Institute of Standards and Technology (NIST): Module-lattice-based key-encapsulation mechanism standard. Technical Report FIPS 203 (August 2023). Accessed: 2024-10-23. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf

[14] National Institute of Standards and Technology (NIST): Module-lattice-based digital signature standard. Technical Report FIPS 204 (August 2024). Accessed: 2024-10-23. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

[15] National Institute of Standards and Technology (NIST): Stateless hash-based digital signature standard. Federal Information Processing Standards Publication NIST FIPS 205, Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.205 . https://doi.org/10.6028/NIST.FIPS.205

[16] National Institute of Standards and Technology (NIST): SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical Report FIPS 202 (August 2015). Accessed: 2024-10-23. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf