

What's the Matter? An In-Depth Security Analysis of the Matter Protocol

Sayon Duttagupta*
COSIC, KU Leuven
Leuven, Belgium
sayon.duttagupta@esat.kuleuven.be

Arman Kolozyan*
MPI-SP & COSIC, KU Leuven
Bochum, Germany
arman@kolozyan.com

Georgio Nicolas*
COSIC, KU Leuven
Leuven, Belgium
georgio.nicolas@esat.kuleuven.be

Dave Singelée
DistriNet (Group T), KU Leuven
Leuven, Belgium
dave.singelee@kuleuven.be

Bart Preneel
COSIC, KU Leuven
Leuven, Belgium
bart.preneel@esat.kuleuven.be

Abstract*—The Matter protocol has emerged as the leading standard for secure IoT interoperability, backed by major vendors such as Apple, Google, Amazon, Samsung, and others. With millions of Matter devices already deployed, its security guarantees are critical to the safety of modern IoT ecosystems. This paper presents the first in-depth analysis of Matter's key establishment protocols: Passcode-Authenticated Session Establishment (PASE) and Certificate-Authenticated Session Establishment (CASE), which are based on the well-studied SPAKE2+ and SIGMA, respectively. We first perform a specification-level security analysis of PASE and CASE, comparing Matter's design choices against the original SPAKE2+ and SIGMA protocols. This reveals weaknesses such as low-entropy setup passcodes, static per-device salts, and weak PBKDF2 parameters. Next, to assess how such weaknesses manifest in practice, we analyse a certified Matter codebase and find that critical safeguards such as limiting failed pairing attempts are not enforced. All identified vulnerabilities were responsibly disclosed and have been patched. We then complement our analysis by formally verifying PASE, CASE, CASE Resumption, and the original SPAKE2+ and SIGMA protocols. In particular, we show in the symbolic model that the security of Matter's lightweight session re-establishment mechanism (CASE Resumption), used for the vast majority of Matter sessions, rests entirely on the secrecy of an ephemeral key generated by an IoT device. Building on all three stages, we construct proof-of-concept attacks and evaluate their feasibility under Matter's own threat model. Our results invalidate several countermeasures specified in Matter's threat model, including protections for threats classified as "High Risk". Overall, our work shows that strong cryptographic building blocks alone are insufficient unless the surrounding protocol and deployment rules are equally rigorously specified and tested at the standard level.

*These authors contributed equally and are listed alphabetically.

1. Introduction

The Internet of Things (IoT) is transforming modern living by bringing automation and connectivity to homes and industries. Yet this growth has also created fragmentation: devices from different manufacturers often rely on incompatible proprietary protocols and ecosystems. To address this, the Matter protocol [1] was developed by the Connectivity Standards Alliance, a consortium comprising prominent industry stakeholders such as Apple, Google, Amazon, Samsung, NXP, and Espressif.

Before Matter, IoT integration was often complex and inconvenient. Users had to manage multiple apps, hubs, and ecosystems to control their devices. Matter addresses this problem by enabling interoperable communication and streamlined commissioning, i.e., pairing, across devices and ecosystems [2], [3]. Its mission to unify the IoT ecosystem also includes a strong focus on security, as Matter is promoted as a security-focused standard [4], with confidentiality, integrity, authenticated communication, and secure commissioning presented as central design goals. This is important because IoT systems have historically suffered from threats such as eavesdropping, unauthorised access, device spoofing, and network intrusion [5], [6], [7], [8], [9].

Matter's security architecture is built around two key establishment protocols: Passcode-Authenticated Session Establishment (PASE) and Certificate-Authenticated Session Establishment (CASE). PASE is used only during device commissioning, while CASE also secures communication between commissioned devices. These protocols are based on the well-studied cryptographic protocols SPAKE2+ [10] and SIGMA [11], respectively. However, Matter does not deploy them in isolation. It embeds them into a larger ecosystem with its own parameter choices, commissioning procedures, and lifecycle assumptions.

While Matter is a relatively new standard, its adoption is growing rapidly: 13,059 unique Matter-certified products are already listed online [12], each potentially manufactured and deployed at a much larger scale. Even a small weakness

in Matter's security design, once exploited, can therefore affect millions of devices worldwide. This makes a thorough security analysis of Matter essential.

We present the first systematic analysis of Matter's key establishment mechanisms, evaluated against Matter's own threat model. By combining practical security assessments with formal verification in the applied pi-calculus, we ask:

> *Do Matter's design choices preserve the security guarantees of the underlying SPAKE2+ and SIGMA protocols, or do they introduce weaknesses that the underlying protocols alone do not exhibit?*

Our findings confirm the latter: we uncover specification-level weaknesses such as low-entropy setup passcodes, weak PBKDF parameters, and static per-device salts. Our formal analysis further reveals that CASE Resumption, the protocol used for the vast majority of Matter sessions, provides significantly weaker security guarantees than a full CASE handshake. We also show that some of the protections Matter relies on, such as limits on failed commissioning attempts, can be bypassed even in certified implementations.

While the Matter documentation alternates between the terms *Matter standard* and *Matter protocol*, for consistency, we use *Matter protocol* to refer to the overall specification. The official Matter specification is available from [1]. At the time of writing, we recommend consulting the "Matter 1.5.1 Core Specification" for technical details. When we refer to Matter's threat model or to a specific threat (e.g., "T102") or countermeasure (e.g., "CM5"), we reference Sect. 13.7 "Threats and Countermeasures" (page 1154) of the 1.5.1 specification. We also reviewed the 1.6 specification, released after submission of this work, and confirmed that all our findings still apply. Throughout the paper, we use *weakness* for specification-level shortcomings, *vulnerability* for implementation-level deviations from the specification.

Methodology. Our analysis proceeds in three stages. First, we perform a specification-level security analysis of PASE (Sect. 4) and CASE (Sect. 5) by studying the Matter specification alongside the original SPAKE2+ and SIGMA designs. This allows us to distinguish properties inherited directly from SPAKE2+ and SIGMA and those affected by Matter's parameter choices and protocol adaptations. Next, we examine whether the specification-level protections are actually enforced in practice. To do so, we analyse a certified Matter codebase for deviations from the specification and evaluate certified devices to observe deployed PBKDF parameters and salt behaviour (Sect. 6). Third, we model PASE, CASE, and CASE Resumption in ProVerif (Sect. 8) to verify whether the protocols satisfy their claimed security properties under idealised cryptographic assumptions. Building on all three stages, we construct proof-of-concept attacks and evaluate their feasibility under Matter's own threat model (Sect. 7). Our findings show that several countermeasures specified by Matter are insufficient in practice, leading us to propose specification-level improvements (Sect. 9).

For readability, all formal verification results are presented together in Sect. 8 rather than interleaved with the rest of the analyses.

Related Work. Matter is widely promoted, implemented, and deployed by numerous IoT vendors worldwide. However, this broad adoption contrasts sharply with the limited scrutiny its design has received from the security and cryptographic research communities. To the best of our knowledge, while the underlying SPAKE2+ protocol has received dedicated security analysis [13], only a handful of academic works have examined Matter's security.

A Master's thesis [14] documents the Matter specifications and outlines potential issues and attack surfaces. It also includes Tamarin models of some protocols, though the analysis did not terminate, rendering the results inconclusive. Another study [15] explores the trust establishment process between devices and controllers, revealing that while controllers verify the credentials of Matter devices, the reverse is not enforced, allowing unauthorised controllers to join the network. This is demonstrated in an attack [16], though under strong assumptions, such as users leaving QR codes on devices post-commissioning, enabling adversaries to scan and pair with them. A separate work [17] exploits vendor-side implementation flaws, gaining unauthorised access via an unenrolled management channel on commissioned devices. Wang et al. [18] show that vendor integrations can introduce design flaws, where controller UIs misrepresent or omit security-critical Matter access-control information (e.g., a list of controllers). More recently, Zohdi and Antonioli [19] identified denial-of-service vulnerabilities in Matter's secure channel establishment. They show that an unauthenticated adversary can disrupt PASE and CASE sessions by exploiting unprotected status report messages.

These studies expose important weaknesses in Matter's trust, availability, UI, and implementation layers. Our work is complementary: we analyse the cryptographic design and security guarantees of Matter's key establishment mechanisms, and focus on the extent to which its claimed guarantees are enforced by cryptographic mechanisms rather than external assumptions.

Contributions. Our main contributions are as follows:

1) **Specification-level security analysis of Matter's key establishment protocols.** We provide a systematic security evaluation of Matter's core key establishment protocols: PASE, CASE, and CASE Resumption. Rather than treating them as isolated uses of SPAKE2+ and SIGMA, we analyse how their security guarantees change once embedded in Matter's commissioning procedures. We identify weaknesses including low-entropy setup passcodes, weak PBKDF parameters, and static salts.
2) **Automated formal analysis of Matter and its underlying protocols.** Using ProVerif, we conduct the first formal analysis of the security properties of PASE, CASE, and CASE Resumption in the symbolic model. While the underlying SPAKE2+ and SIGMA protocols have received prior formal treatment in the computational model, the security of Matter's specific protocol variants has not previously been verified in the literature. As a complementary contribution, we also formally analyse SPAKE2+ and SIGMA in a similar way. Most notably,

we show that CASE Resumption provides significantly weaker guarantees than a full CASE handshake. All models are fully documented, allow automatic reproduction, and serve as a foundation for researchers who wish to carry our results further.

3) **Identification of implementation-level vulnerabilities.** We identify multiple deviations from the Matter specification in a certified codebase, including failures to correctly enforce commissioning restrictions and cryptographic input validation. These deviations introduce exploitable vulnerabilities despite the presence of the corresponding countermeasures in the Matter specification. All vulnerabilities were responsibly disclosed and subsequently patched.
4) **Attacks and empirical validation.** We present a passcode recovery attack from leaked device material and a controller-in-the-middle attack during recommissioning, and evaluate their feasibility with benchmarks under Matter's own threat model. We also test certified devices to measure real-world parameters, confirming that weak configurations occur in deployed devices.
5) **Analysis of Matter's countermeasures.** Based on our findings, we show that specific countermeasures in Matter's *Threats and Countermeasures* list are insufficient in practice, including claims that pre-computation attacks are infeasible. We propose protocol-level improvements to address the identified weaknesses and derive broader lessons for large-scale cross-vendor IoT standards.

Artefacts. We provide an artefact bundle at https://github.com/KULeuven-COSIC/whats-the-matter. It includes source code for our attacks, benchmarks, documented formal models, and ProVerif output logs.

2. Foundations

This section provides the foundational concepts necessary to understand Matter's security architecture. We describe its core components, as well as the commissioning process that onboards new devices. Furthermore, in Appendix B, we give an overview of the crypto suites used in Matter.

2.1. Matter's Building Blocks

Matter's infrastructure comprises the following core components [1].

Devices and nodes. A *device* is a physical unit, such as a lightbulb or thermostat. A device may host one or more *nodes*, which are logical, addressable entities that run the Matter protocol stack and expose device functionality.

Fabrics. A *fabric* is a group of Matter devices under a shared trust root, usually one per ecosystem such as Apple's or Google's. A device can join several fabrics at once, so both can control the same lightbulb.

Commissioners and controllers. *Commissioners* onboard new devices, verify their credentials, and provision them with operational certificates. This could be a phone app scanning a lightbulb's QR code, after which that phone (or an associated smart speaker) acts as a *controller* that operates the device within the fabric by sending it commands.

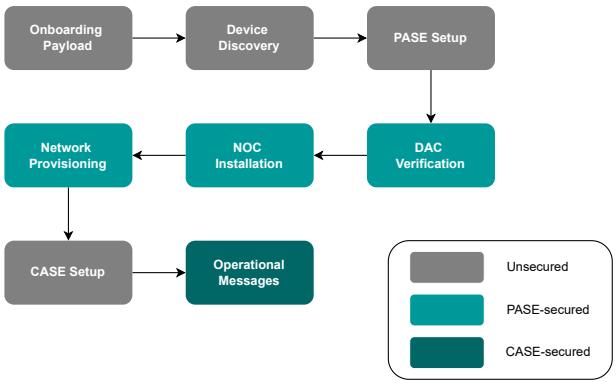

Figure 1. Overview of the Matter commissioning process, which advances through escalating security levels: initial unsecured communication for PASE setup, PASE-secured exchange for DAC and NOC provisioning, and CASE-secured communication for operational use.

2.2. Commissioning in Matter

The commissioning process in Matter is a multi-phase, secure procedure for onboarding new devices (commissionees) into a fabric, shown in Fig. 1. We describe its individual steps below.

Passcode. The device's onboarding payload, shared out-of-band with the commissioner (e.g., via QR code, NFC, or manual entry), contains metadata such as the device version, vendor ID, and product ID, as well as an 8-digit setup passcode that acts as the shared secret for Passcode-Authenticated Session Establishment (PASE): the commissioner proves knowledge of the passcode, while the commissionee proves possession of verifier material derived from it. Matter supports static and dynamic passcodes. The latter are generated per session via a cryptographically secure pseudorandom number generator, but as many constrained devices lack displays or speakers, static passcodes remain the most common choice.

Device discovery and initialisation. The process begins with the commissionee announcing its readiness to set up a secure session, typically using Bluetooth Low Energy (BLE) or IP-based discovery. The commissioner discovers the device, initiates communication, and obtains the parameters needed to start PASE.

PASE session establishment. Using the shared passcode, the commissioner and commissionee execute PASE to establish a passcode-authenticated secure channel. This channel protects subsequent commissioning messages until CASE is executed. After PASE completes, the commissionee arms the fail-safe timer, which limits the commissioning window to 60 seconds.

Attestation and validation. The commissioner authenticates the device by verifying its Device Attestation Certificate (DAC) and Product Attestation Intermediate (PAI)

certificate, both issued by trusted certificate authorities. This step ensures that only legitimate devices can join the fabric.

Operational certificate assignment. After successful attestation, the device generates an operational key pair and sends a Certificate Signing Request (CSR) to the commissioner. The commissioner then forwards it to the fabric's certificate authority, which issues a Node Operational Certificate (NOC). The NOC becomes the device's cryptographic identity within the fabric and is essential for communicating with other devices in the network. Further details on NOCs are provided in Appendix C.

Network provisioning. The commissioner configures the device's network parameters, such as IP addresses or Wi-Fi credentials, and instructs it to join the target network.

CASE session establishment. With the NOC installed, the commissioner and commissionee use their operational certificates to establish a Certificate-Authenticated Session Establishment (CASE) channel. This ensures mutual authentication and secures all subsequent communication within the fabric with confidentiality and integrity protection. Once completed, the device is fully integrated, the fail-safe timer is disarmed, and it can now securely interact with other Matter-enabled nodes.

Recommissioning. A device can be recommissioned using one of two methods, decided at manufacturing time. The *Basic Commissioning Method* (BCM) reuses the device's original passcode. In the *Enhanced Commissioning Method* (ECM), an existing controller generates a new random passcode, shared on-screen or via audio. A factory reset always restores the original passcode.

3. Threat Model

This section defines the security goals, assumptions, and adversaries used throughout our analysis. Our threat model is based on Matter's own *Threats and Countermeasures* framework (Sect. 13.7, p. 1154 of the 1.5.1 specification). In other words, we do not introduce an independent threat model, but solely adopt Matter's own threat landscape and evaluate whether its specified countermeasures are sufficient in practice.

3.1. Security Goals

Matter aims to establish authenticated and confidential communication channels between IoT devices during both commissioning and subsequent operational use. These guarantees rely on the secrecy of setup passcodes, certificate chains, and keys used during commissioning.

The Matter specification discusses security objectives only at a high level (e.g., "secure commissioning") without formally defining the precise security guarantees. To enable a systematic analysis, we therefore map Matter's stated or implied objectives to established cryptographic security notions adapted from [11], [13], [20]:

- **Confidentiality.** A value exchanged between two honest parties is confidential if it cannot be learned by any unauthorised third party.
- **Mutual Authentication.** If a protocol execution successfully completes between two honest parties, then each party is assured of the identity of the peer with which it established the session, according to the authentication mechanism intended by the protocol.
- **Identity Protection.** A protocol provides identity protection if protocol participants do not reveal their identities to unauthenticated parties prior to establishing sufficient trust in the peer.
- **Forward Secrecy.** Session keys established prior to the compromise of long-term keys remain confidential.
- **Session Independence.** The compromise of ephemeral state or session keys from one protocol execution does not compromise the security of unrelated sessions.
- **Resistance to Offline Dictionary Attacks.** An adversary cannot verify secret setup credentials without interaction with an honest party.

3.2. Assumptions

We assume that honest parties follow either the Matter specification or officially certified Matter codebases. We further assume:

- **Sufficient Entropy.** Devices have access to sufficient entropy for the generation of cryptographic randomness. This may be optimistic: a DRBG is only as strong as its seed, and whether constrained devices reliably produce good seeds remains an open question.
- **Secure Primitives.** Matter relies on cryptographic primitives aligned with NIST guidelines, including AES-CTR-DRBG, SHA-256, HMAC-SHA-256, AES-CCM, and PBKDF2. Further details are provided in Appendix B. Unless stated otherwise, we assume these primitives are cryptographically secure and correctly implemented.
- **Secure NOCs.** Node Operational Certificate (NOC) chains are correctly issued, transferred, and validated by honest parties.

3.3. Adversaries

We consider the following adversaries, all of which are grounded in capabilities described in Matter's own *Threats and Countermeasures* model:

- **Passive Network Adversary.** The attacker can observe, record, and analyse traffic exchanged between honest Matter participants.
- **Active Network Adversary.** The attacker can inject, replay, modify, delay, or suppress messages.
- **Malicious Commissioner.** The attacker controls a commissioner device.
- **Adversary with Compromised State.** In selected scenarios, the adversary has access to secret state such as setup passcodes and verifier material (as assumed in offline brute-force threat *T102* and passcode-based pairing threats *T173–T175*).

We exclude physical tampering with devices.

4. PASE Setup Analysis

The Passcode-Authenticated Session Establishment (PASE) protocol is the first cryptographic gate in Matter's

commissioning process. It is executed between a commissioner and an uncommissioned device, called the commissionee, before device attestation, network provisioning, operational certificate installation, and CASE establishment. PASE therefore bootstraps the secure path over which the device is admitted into a Matter fabric.

PASE is based on SPAKE2+, a Password-Authenticated Key Exchange (PAKE) protocol that allows two parties to derive a shared secret from a low-entropy password [10]. SPAKE2+ is an *augmented* PAKE, meaning that the verifier side does not need to store the password itself, but only verifier material derived from it. In Matter, the shared low-entropy secret is the setup passcode, the prover role is played by the commissioner, and the verifier role is played by the commissionee.

In practice, the majority of Matter devices perform this setup using a static 8-digit passcode, as dynamic passcodes require a display or similar output interface. For this case, the passcode for each device is generated during the manufacturing process. A QR code encoding the passcode and additional metadata is printed and provided with the device. The device does not store the passcode directly. Instead, following Matter's PASE terminology, the passcode is first expanded via PBKDF2 into two scalars w_0 and w_1. The device then stores w_0 and $L = w_1 \cdot P$, where P is the generator of the elliptic curve group.

We discuss the original SPAKE2+ protocol in Appendix F.1. There are two layers of deviations between SPAKE2+ and PASE. First, Matter adopts the second draft of SPAKE2+, which differs from the final RFC in terminology and key derivation (detailed in Appendix D). Second, PASE introduces its own changes on top of that draft. A full list of these PASE-specific deviations is provided in Appendix F.3. We focus on PASE itself in the rest of this section.

4.1. PASE Protocol Overview

As a step towards establishing the keys, the commissioner needs to prove knowledge of the passcode to the commissionee. It is assumed that the commissioner has obtained the passcode through some out-of-band method (such as scanning a QR code), and that the commissionee has the corresponding (w_0, L) pair pre-installed on the device prior to protocol initiation.

The flow of PASE is summarised below and illustrated in Fig. 2. The protocol relies on a cyclic group G of order $p \cdot h$ (where p is a large prime and h is the cofactor) with generator P. Two public group elements M and N are used to blind the key exchange.

1) **PBKDF parameters exchange and key generation**
   a) The commissioner sends a *PBKDFParamRequest* to the commissionee.
   b) The commissionee responds with a *PBKDFParamResponse* containing the salt and iteration count.
   c) The commissioner computes the values w_0 and w_1 using the shared passcode and the received parameters:

$$w_{0s} \parallel w_{1s} = \text{PBKDF}(\text{passcode}, \text{salt}, \text{iter})$$
$$w_0 = w_{0s} \pmod p$$
$$w_1 = w_{1s} \pmod p$$

   The commissionee already holds w_0 and $L = w_1 \cdot P$ (pre-computed during manufacturing). Note that, since the verifier is derived before any commissioner identity is known, Matter feeds only the passcode into the PBKDF, rather than also binding the participant identities as the SPAKE2+ draft allows.

2) **SPAKE2+**
   a) The commissioner generates a random scalar $x \in [0, p-1]$, computes the ephemeral public key $pA = x \cdot P + w_0 \cdot M$, and sends pA to the commissionee.
   b) The commissionee checks that pA is valid, then generates a random scalar $y \in [0, p-1]$ and computes $pB = y \cdot P + w_0 \cdot N$. Next, it calculates the shared secrets:

$$Z = h \cdot y \cdot (pA - w_0 \cdot M) = h \cdot y \cdot x \cdot P$$
$$V = h \cdot y \cdot L = h \cdot y \cdot w_1 \cdot P$$

   The commissionee computes a confirmation hash and sends pB to the commissioner.
   c) The commissioner validates pB, then computes the same shared secrets:

$$Z = h \cdot x \cdot (pB - w_0 \cdot N) = h \cdot x \cdot y \cdot P$$
$$V = h \cdot w_1 \cdot (pB - w_0 \cdot N) = h \cdot w_1 \cdot y \cdot P$$

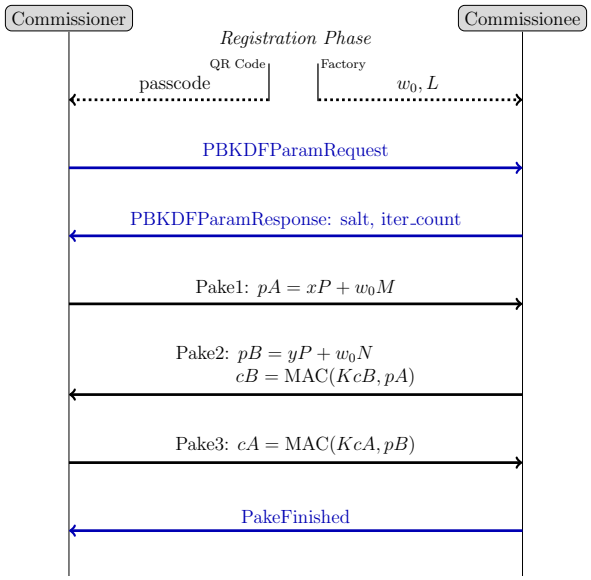

Figure 2. PASE overview. The registration phase is represented with dotted lines: the device stores verifier material (w_0, L) during manufacturing, and the commissioner obtains the passcode by scanning the device's QR code. Messages added by Matter around SPAKE2+ are highlighted in blue.

3) **Key derivation and confirmation.** Both parties derive the session keys from the protocol transcript TT, which is defined as follows:

$$TT = \text{len}(A) \parallel A \parallel \text{len}(B) \parallel B \parallel \text{len}(M) \parallel M \parallel \text{len}(N) \parallel N \parallel \text{len}(pA) \parallel pA \parallel \text{len}(pB) \parallel pB \parallel \text{len}(Z) \parallel Z \parallel \text{len}(V) \parallel V \parallel \text{len}(w_0) \parallel w_0$$

The secrets Ka and Ke are derived as follows:

$Ka \parallel Ke = \text{Hash}(TT)$

Each key is half the length of the hash output. From Ka, confirmation keys KcA and KcB are derived using a KDF:

$KcA \parallel KcB = \text{KDF}(\text{nil}, Ka, \text{“ConfirmationKeys”})$

The confirmation keys are used to generate and verify key confirmation messages, ensuring both parties agree on the derived secrets before proceeding. Specifically, the prover sends $cA = \text{MAC}(KcA, pB)$ and the verifier sends $cB = \text{MAC}(KcB, pA)$. Once keys are confirmed, Ke becomes the authenticated shared secret for all future encryption. It is important to note that Ka and its derived keys serve no purpose beyond key confirmation and should be securely erased from memory.

4.2. Security of PASE

We now examine the security of PASE as deployed in Matter. SPAKE2+ itself is proven to provide strong security guarantees under its intended assumptions [13]. However, once integrated into the Matter ecosystem, several of Matter's design choices weaken the commissioning security that PASE inherits from it. We discuss each concern in this subsection. In Sect. 7, we present attacks that exploit the weaknesses discussed in this section. Finally, our proposed recommendations are covered in Sect. 9.

Low-entropy setup credentials. Matter setup passcodes are 8-digit random numbers, giving at most 10^8 candidate values, as specified by Matter's passcode entropy requirement (*CM5*). This does not by itself imply that PASE is broken: PAKE protocols are specifically designed to use low-entropy secrets while preventing offline guessing from passive transcripts. However, the small passcode space makes Matter heavily dependent on online enforcement mechanisms such as retry limits (*CM100*, targeting online brute-force threats *T101* and *T112*). As we show in Sect. 6, even certified implementations do not always enforce these limits correctly.

Weak PBKDF parameters. Matter's PBKDF countermeasure (*CM99*) requires devices to utilise multiple hashes in the PBKDF. Sect. 3.9 of the specification sets the permitted iteration count between 1,000 and 100,000. This range is significantly lower than the currently recommended minimum of 600,000 iterations for *PBKDF2-HMAC-SHA256* in OWASP guidance [21]. The consequence is not that PBKDF2 immediately breaks PASE, but that the cost of exhaustive search over Matter's small passcode space is reduced whenever an attacker obtains material that allows offline verification of guesses.

Static salts and verifier material. As discussed in Sect. 2.2, most devices use a static passcode, from which the on-device verifier (w_0, L) is derived via PBKDF2 with a specific salt. Since the verifier is pre-computed during manufacturing, the salt stays fixed for the device's lifetime: any other salt would yield a verifier that no longer matches the passcode. Moreover, any party on the network can retrieve the salt through an unauthenticated *PBKDFParamRequest*. An attacker can therefore pre-compute a table mapping every candidate passcode to its w_0, and recover the passcode by lookup once w_0 leaks from a compromised commissioner (Sect. 7.1). We discuss mitigations in Sect. 9.

Missing group membership checks. During PASE, both parties exchange ephemeral public keys (pA and pB). SPAKE2+ requires each party to verify that the received key is a valid element of the prime-order group, and to abort otherwise [10], [22]. Without this check, an adversary can submit a point lying on a different, weaker curve. The secret the receiver derives is then confined to a small set of values, leaking part of its private scalar. Repeating this with different curves against the same long-term scalar recovers it in full. Such invalid-curve attacks are a well-known class of attack against elliptic-curve protocols [23], [24].

Matter fixes a single curve, secp256r1, whose prime point count leaves no small subgroups [25], so validation reduces to checking that the received point lies on the curve and is not the identity. However, the specification does not mandate this check, and we found that the official codebase omits it in practice (Sect. 6). To assess whether invalid-curve attacks apply, we consider every scalar multiplication in PASE involving a received point: the attack requires the targeted scalar to act on a point the adversary can place on a curve of its choice. w_0 never meets this condition, as it only multiplies the fixed elements M and N. The remaining multiplications, by x and w_1 on the commissioner side and by y on the commissionee side, act on the differences $pB - w_0 \cdot N$ and $pA - w_0 \cdot M$, respectively. For an adversarial pB, the curve on which $pB - w_0 \cdot N$ lies is determined by w_0: without it, the adversary can neither choose that curve nor confine the result to a small subgroup (the same holds for pA). Hence neither the ephemeral scalars nor w_1, whose recovery would expose the setup passcode by offline search, can be extracted this way. An adversary who does know w_0 recovers the passcode more cheaply through the attack of Sect. 7.1. We thus consider the missing mandate a latent specification gap rather than an exploitable flaw.

No identity authentication. PASE authenticates only knowledge of the setup passcode. The commissionee hence cannot distinguish a legitimate commissioner from any other passcode holder [15]. Conversely, the specification permits the commissioner to continue even when device attestation fails. Sect. 7.2 shows how these gaps enable a controller-in-the-middle attack.

PBKDF on the commissioner side. Finally, while not directly a security concern, it is unclear why the commissioner is required to apply a PBKDF to the passcode. An attacker with the passcode can apply the same PBKDF to derive w_0 and w_1. Why does Matter not include w_0 and w_1 directly in the QR code? One rationale could be to keep the

manual passcode entry short when no QR is available. This interpretation is supported by the Matter specification: "For example, a device may not require the *Crypto_PBKDF()* primitive, as values based on this operation could in some instances be pre-computed and stored during the manufacturing process of the device."

5. CASE Setup Analysis

After a successful PASE session, the Certificate-Authenticated Session Establishment (CASE) protocol is used to provide mutual authentication between peer nodes and to generate keys securing subsequent communications. Unlike PASE, which bootstraps trust from a setup passcode, CASE relies on Node Operational Certificates (NOCs) and operational private keys installed during PASE.

CASE is based on SIGMA, a family of key-exchange protocols designed for secure, authenticated Diffie-Hellman key exchange using digital signatures and message authentication codes (MACs). SIGMA stands for "SIGn-and-MAc", reflecting its use of both primitives. The protocol forms the cryptographic backbone of Internet Key Exchange (IKE), the standardised key-exchange protocol for IPsec [11]. We detail the workings of SIGMA in Appendix F.2.

5.1. CASE Protocol Overview

As discussed in Sect. 2.2, during commissioning the device receives a Node Operational Certificate (NOC) over the PASE-secured channel. The NOC, detailed in Appendix C, binds the node to a fabric and provides the public key material used for authentication in CASE.

At a high level, CASE proceeds as follows. The initiator and responder exchange fresh ephemeral elliptic-curve public keys and derive a shared Diffie-Hellman secret. The responder then authenticates itself by sending its operational certificate together with a signature over the exchanged public keys and certificate, all encrypted under keys derived from the shared secret. The initiator verifies the responder's certificate chain and signature, then replies with its own certificate and signature under similarly derived protection. If both verifications succeed, the two parties derive shared session keys for protecting subsequent Matter messages.

The Matter specification describes CASE as mirroring "the SIGMA protocol," without specifying a variant. Strictly speaking, SIGMA is a family of protocols [11], and CASE specifically follows the SIGMA-I variant. In particular, the initiator (e.g., the commissioner) only reveals its identity to the responder (e.g., the commissionee) after verifying the responder's identity. Matter's concrete CASE message flow includes additional protocol fields, such as random values, session identifiers, and a final status report. These differences are summarised in Appendix F.4.

Fig. 3 illustrates the protocol flow of CASE. In this figure:

- g^x and g^y are the ephemeral Diffie-Hellman public keys of the initiator and responder, respectively.
- $\mathrm{Sig}_R(g^x, g^y, \mathrm{NOC}_R)$ and $\mathrm{Sig}_I(g^y, g^x, \mathrm{NOC}_I)$ are digital signatures over both ephemeral public keys and the sender's operational certificate.

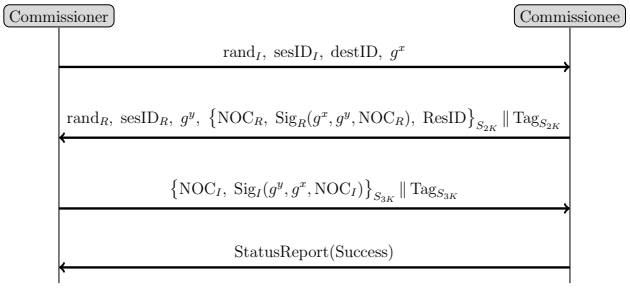

Figure 3. Protocol flow of the Certificate Authenticated Session Establishment (CASE) protocol in Matter.

- $\{\cdots\}_{S_{2K}}$ and $\{\cdots\}_{S_{3K}}$ denote authenticated encryption with keys derived from the shared secret.
- $\mathrm{Tag}_{S_{2K}}$ and $\mathrm{Tag}_{S_{3K}}$ are AEAD authentication tags appended to the encrypted blobs.

Key derivation. Matter names the three CASE messages *Sigma1–Sigma3*, and S_{2K} and S_{3K} are the AEAD keys protecting the second and third. Both, and the session keys, derive from the Diffie-Hellman shared secret $Z = g^{xy}$ together with the *Identity Protection Key* (IPK), a symmetric key shared by all members of a fabric. Writing H_i for the hash of the first i messages and r_R for the responder's random value:

$$S_{2K} = \mathrm{KDF}(Z,\ \mathrm{IPK} \| r_R \| g^y \| H_1,\ \text{"Sigma2"})$$
$$S_{3K} = \mathrm{KDF}(Z,\ \mathrm{IPK} \| H_2,\ \text{"Sigma3"})$$
$$\mathit{SessionKeys} = \mathrm{KDF}(Z,\ \mathrm{IPK} \| H_3,\ \text{"SessionKeys"})$$

The IPK enters the salt of every key, so an adversary who learns Z cannot derive any CASE key without it.

5.2. CASE Resumption

Matter also specifies CASE Resumption, a mechanism for re-establishing a secure session without executing a full CASE handshake. The motivation is efficiency: resumption avoids the costly signature generation and verification steps of CASE, which is particularly beneficial for resource-constrained IoT devices.

During a full CASE execution, the responder generates a resumption identifier (*ResID*) and sends it encrypted to the initiator. Both parties store it together with the session's shared secret Z. To resume later, the initiator sends a resumption request with the stored *ResID* and a fresh random value r_I. If the responder finds matching resumption state, both parties derive new session keys from the stored secret and the public nonces, without Diffie-Hellman operations, certificate exchange, or signature verification:

$$\mathit{SessionKeys} = \mathrm{KDF}(Z, r_I \| \mathit{ResID}, \text{"SessionResumptionKeys"})$$

While this improves performance, it also affects the trust assumptions underlying the session. A full CASE handshake authenticates a fresh ephemeral key exchange using long-term operational credentials (NOCs and their private keys). A resumed session, by contrast, relies entirely on the continued secrecy of the stored resumption state. As a result,

the security of resumed sessions cannot be inferred from the security of the full CASE handshake alone.

5.3. Security of CASE

Use of AEAD instead of separate MAC. As detailed in Appendix F.4, in standard SIGMA each party sends a signature alongside a separate MAC over its identity, using distinct keys for MAC and encryption. Matter replaces the separate MAC and encryption with a single AEAD operation: each party first signs the ephemeral public keys and its certificate, then encrypts the signature and certificate under AEAD. The AEAD authentication tag implicitly provides the identity binding that SIGMA's separate MAC was designed to achieve [11]. Since AEAD is the standard modern primitive for combining encryption and authentication, this is a sound design choice.

Choice of SIGMA-I and identity exposure. The Matter specification does not explain why CASE follows the SIGMA-I variant rather than SIGMA-R. In SIGMA-I, the initiator's identity is protected against active attackers, while the responder's identity is protected only against passive eavesdroppers. A likely rationale is that the commissioner (initiator) typically holds more privileges in the network than the commissionee (responder), making its identity more valuable to protect. However, the specification itself acknowledges a limitation: "Since there are no trust mechanisms employed for commissioners advertising themselves, commissionees may provide commissioner selection choices to the user that are from malicious entities masquerading as commissioners." This means that a malicious commissioner can actively probe commissionees and learn their identities, since SIGMA-I does not protect the responder's identity against active attackers.

Security downgrade in session resumption. Our formal analysis reveals that session resumption significantly weakens the security guarantees compared to a full CASE handshake. This lowers what an attacker needs to control the device and decrypt its traffic, which is the impact Matter's threat model attaches to compromised key material. The specific findings and a proposed patch backed by a formal model are detailed in Sect. 8.

6. Implementation Analysis

To assess whether Matter's specification-level protections are enforced in practice, we examined the official open-source JavaScript Matter Software Development Kit (SDK), *matter.js*[*]. This SDK is explicitly marked as compliant with Matter version 1.5.1 in the Matter Handbook[†], and is part of the official Matter GitHub repository[‡]. We identified four vulnerabilities in the SDK that deviate from the specification and undermine critical commissioning protections. All these code-level issues were responsibly disclosed to the developers, who patched them promptly. The missing group-membership check remains under investigation at the specification level. Appendix A details our disclosure process.

Failure to enforce commissioning attempt limits. Matter's countermeasure *CM100* states that a "device exits commissioning mode after 20 failed commissioning attempts," cited explicitly for threats *T101* and *T112*, which concern online brute-force attacks on the setup passcode. We found that this limit is not effectively enforced in the SDK[§]. Although the SDK logs the message "Maximum number of PASE pairing errors reached, cancelling commissioning" once the threshold is reached, it continues to accept and process new pairing requests. The internal error counter even continues incrementing beyond the threshold (e.g., 21/20, 22/20, etc.), effectively nullifying the intended protection.

Failure to enforce the commissioning window. Matter's specification recommends a 60-second commissioning window (Sect. 5.5). We found that the SDK does not enforce this timeout, allowing an attacker to terminate failed attempts early, without waiting for the device's status report. Without a timeout, an attacker who also bypasses the retry limit can issue attempts continuously, reducing the per-attempt cost of online brute-force.

Failure to enforce commissioning exclusivity. The Matter specification requires that "when a Commissioner is either in the process of establishing a PASE session with the Commissionee or has successfully established a session, the Commissionee shall not accept any more requests for new PASE sessions until session establishment fails or the successfully established PASE session is terminated on the commissioning channel." We found that the SDK did not reject new PASE session attempts during ongoing session establishment[¶]. This enables an attacker who knows the passcode to passively wait for a legitimate user to initiate commissioning, then interrupt the process and take over the session with its own controller, without physical access to the device. This directly undermines Threat *T15* ("Commission an uncommissioned Node without physical access to Device") and its countermeasure *CM3*, which mandates that "Commissioning is started with some form of physical user interaction (e.g. power cycle or button press)."

Missing group membership checks. The group-membership check discussed in Sect. 4.2 was also absent from the codebase.

7. Attacks on Matter's Protocols

We now show how the specification-level weaknesses discussed in Sect. 4 and the implementation vulnerabilities identified in Sect. 6 can be exploited in practice. The attacks presented in this section operate under assumptions described in Matter's own threat model and challenge the security claims of several countermeasures that Matter classifies as "High Risk". Our goal is not to claim that every Matter device is immediately exploitable, but to show that certain protocol design choices, strengthened by implementation

[*]https://github.com/project-chip/matter.js/

[†]https://handbook.buildwithmatter.com/development/

[‡]https://github.com/project-chip

[§]https://github.com/project-chip/matter.js/issues/1894

[¶]https://github.com/project-chip/matter.js/issues/1896

shortcomings, can make specific attacks practical. Based on these findings, we propose spec-level mitigations in Sect. 9.

We focus on two attacks. The first recovers the setup passcode from PASE verifier material, showing that Matter's low passcode entropy and long-lived commissioning parameters can make post-compromise attacks practical. The second is a controller-in-the-middle attack during commissioning, showing that an attacker who knows the passcode can interpose between a legitimate commissioner and the device during the commissioning window, influencing the device's operational identity, fabric membership, and access-control state.

7.1. Post-Compromise Passcode Recovery

As described in Sect. 4, Matter's setup passcode is an 8-digit value, from which the PASE verifier material (w_0, L) is derived via PBKDF2 using a static salt. Both (w_0, L) and the salt are fixed at manufacturing time and remain unchanged for the lifetime of the device. The salt can be retrieved by anyone on the network through an unauthenticated *PBKDFParamRequest*.

Matter's threat model lists threat *T102* as high-severity and claims that a pre-computation attack, where an attacker who has access to w_0 and L attempts to recover the passcode, is not possible. We show that this statement does not hold: the passcode can even be recovered from w_0 alone, without needing L.

Lifecycle. The static and publicly retrievable salt lets an attacker pre-compute a table mapping every candidate passcode to its w_0 value. Once w_0 is compromised, a single lookup recovers the passcode. Its lifecycle then depends on the recommissioning method (Sect. 2.2). Under BCM, the device reuses its original passcode, so the recovered one stays valid. Under ECM, a new passcode is issued, but a factory reset restores the original.

Assumptions. This attack assumes the following:

- The attacker has access to w_0. Note that this is a strictly weaker assumption than threat *T102* described in Matter, which assumes the attacker has both w_0 and L.
- The salt used in the PBKDF is known to the attacker. This is also not a hard assumption, since the PBKDF parameters (salt and number of iterations) can be obtained by the attacker via a simple, unencrypted request to the device.

Exploited weaknesses. The root cause is the size of the passcode space: 10^8 candidates are few enough to exhaust offline once w_0 leaks. The static salt only determines *when* that work happens: since it never changes, the table can be built before w_0 is obtained, reducing recovery to a single lookup. With a fresh salt, the attacker would simply run the same exhaustive search after the leak instead of in advance. Building the table requires all 10^8 derivations, about 10h on a single core at 1,000 iterations (Table 2). Searching once w_0 is known instead stops at the matching candidate, halving the expected time to about 5h.

Computational cost. Building the full table takes about 10h on a single core at the 1,000 iterations used by one of our test devices. On 128 cores this falls to 5 minutes, and to 8h at Matter's permitted maximum of 100,000 iterations. Once built, a lookup takes 33s with a naive linear scan and milliseconds with a hash map. Appendix E reports the full measurements.

Implementation and validation. We implemented the attack using the official Matter software development kit (SDK)[‖] and its `lighting-app` example application. The code is available as part of our artefact. Furthermore, to verify that production devices indeed use static salts and low iteration counts, we tested two Matter-certified devices: the TP-Link Tapo P100M (smart plug) and the Nous P3 (smart bulb). For each device type, we acquired multiple units to verify consistency across production batches. Both manufacturers indeed use static and device-specific salts. The Nous P3 uses the minimum permissible iteration count of 1,000 and the TP-Link Tapo P100M uses 15,000 iterations.

Impact. This attack invalidates Matter's own security claim: threat *T102* states that pre-computation attacks on the passcode are infeasible, yet we demonstrate that the passcode can be recovered under weaker assumptions. Under the commissioning conditions described above, the recovered passcode can enable an attacker to commission and control the device, including access to sensitive data. This impact is explicitly listed in Matter's threat model for this attack as "Control of Device and access to sensitive Device data (e.g. IP Camera traffic)." Moreover, the recovered passcode can be fed directly into the controller-in-the-middle attack we cover in the next subsection. We discuss mitigations in Sect. 9.

7.2. Controller-in-the-Middle Attack

We next consider a Controller-in-the-Middle (CitM) attack. Matter's threats *T173–T175* assume an attacker with knowledge of the passcode who attempts to pair with or re-commission a device, and threat *T15* considers commissioning without physical access. The attack exploits the two limitations described in Sect. 4.2: the commissioner is never authenticated, and commissioning proceeds even when device attestation fails. Together, these allow a malicious commissioner to interpose itself between a legitimate commissioner and the device.

The attack is depicted in Fig. 4 and proceeds as follows:

- **T0.** A lightbulb with a QR code to be commissioned.
- **T1.** An honest commissioner pairs the device and learns its passcode.
- **T2.** That commissioner is later removed, but keeps the passcode and remains on the local network, e.g., a former housemate or an offboarded employee.
- **T3.** An honest commissioner re-commissions the device, using BCM or after a factory reset. The attacker acts as the actual Matter device with an invalid certificate and completes the commissioning process with the new honest commissioner. At the same time, it commissions with the real device using the retained passcode.

Assumptions. This attack assumes the following:

[‖] https://github.com/project-chip/connectedhomeip

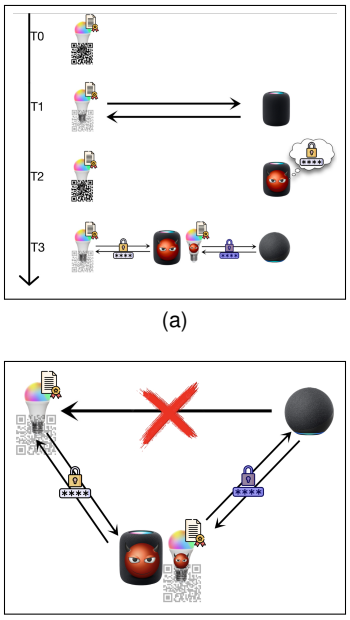

Figure 4. Controller-in-the-Middle Attack. (a) Attack timeline from initial commissioning (T1) to interposition (T3). (b) During interposition (T3), the attacker completes PASE with the real device while presenting a fake device to the legitimate commissioner. A faded QR code indicates that the device is already commissioned, and different lock colours represent distinct session keys.

- The attacker has prior knowledge of the passcode. If the attacker only possesses w_0, then the attack described in Sect. 7.1 can first be leveraged to recover the passcode.
- The attacker can impersonate a Matter device. As outlined in the specification, the commissioner may proceed with commissioning even if device attestation fails (i.e., when the commissionee has an invalid certificate).
- The malicious controller can remain unnoticed by the user. Recent work has shown that the management interfaces of legitimate controllers can misrepresent or omit access-control information, making it harder for users to detect that an unauthorised controller has joined the network [18].

Exploited weaknesses. The commissioner is never authenticated, and commissioning proceeds even when device attestation fails.

Performance. Once the attacker has obtained the passcode, the attack requires no additional computation beyond running two standard PASE handshakes concurrently. After interposition, the attacker can relay messages transparently without disrupting the normal operation of the device.

Implementation and validation. While we did not evaluate CitM on commercial devices, Matter's implementation exposes a `--bypass-attestation-verifier` flag, for which "the commissioning will continue in case of the attestation verification failure" [26]. A key prerequisite of the attack is therefore supported in the official codebase.

Impact. Matter's countermeasures for threats T173–T175 rely on the assumption that passcode knowledge alone is insufficient to compromise commissioning. This attack shows otherwise: under the assumptions described above, an attacker who knows the passcode can interpose itself as a man-in-the-middle, undermining the trust model that Matter's commissioning security is built on. A successful CitM attack allows the malicious actor to intercept and manipulate all communication between the device and the controller. The attacker can then eavesdrop on sensitive information and relay commands from the fake device to the real device, maintaining control without detection under the same assumptions. We discuss mitigations in Sect. 9.

8. Formal Verification of Matter's Protocols

We complement our specification analysis with formal verification to determine whether Matter's key establishment protocols satisfy their claimed security properties under idealised cryptographic assumptions. We conduct our formal analysis using *ProVerif* [27], an automated symbolic verification tool for cryptographic protocols. *ProVerif* operates in the Dolev-Yao adversarial model, where cryptographic primitives such as encryption, hashing, and signatures are treated as ideal black boxes. In this model, an encryption cannot be broken without the key, a hash cannot be inverted, and parameters such as the PBKDF iteration counts are implicitly assumed to be sufficient. The adversary fully controls the network and can intercept, drop, modify, replay, or forge messages.

Protocol participants are modelled as *processes* in the applied pi-calculus. Each process describes the sequence of steps a party performs, including sending and receiving messages over *channels* that represent real-world communication links. Given such a model, *ProVerif* automatically checks whether specified security properties (expressed as *queries*) hold or can be violated by the adversary. We refer the interested reader to the *ProVerif* manual [27] for a comprehensive introduction to the tool.

8.1. Verification Methodology

Scope. We model PASE, CASE, and CASE Resumption, and verify the security properties they claim to provide. As supplemental material, we also model the underlying protocols they are based on: SPAKE2+ for PASE and SIGMA-I for CASE. We additionally provide tweaked model variants that each leak a single secret: w_0, supporting the attack of Sect. 7.1, and either ephemeral key, supporting our CASE Resumption analysis. All models are fully documented and included in the artefact package referenced in Sect. 1.

Modelling process. We translate each protocol from its textual specification and pseudocode into the applied pi-calculus. We use a standard library of cryptographic primitives that provides all required building blocks. All communications take place over a single public channel that the adversary can read from or write to. For CASE, the honest parties are an Initiator and a Responder. For PASE, they are a Prover and a Verifier. We allow an unbounded number of concurrent sessions to capture interleaving attacks.

Adversary model. We begin each analysis with the default *ProVerif* active attacker, who controls the network. We then progressively stretch the adversary's capabilities by leaking additional material (e.g., setup credentials, seeds, and keys) to identify precisely which compromises break which properties.

Events and phases. We use two key mechanisms in our models. First, *events* are labels that we insert at specific points in a protocol process, so that *ProVerif* can track which protocol steps each party has completed and with which values. For example, we could place the following event at the end of the initiator's protocol code:

```
event initiator_end(initiator_noc, gx,
    initiator_random);
```

When the initiator reaches this point, the event is triggered. This specific event is bound to the initiator's identity (`initiator_noc`), public key share (`gx`), and randomness (`initiator_random`) used in that session. We can then query *ProVerif* to determine whether certain events have been triggered and under what conditions. For instance, we can ask: "whenever the initiator records `initiator_end`, was there a corresponding event from the responder with matching parameters?" If so, we can deduce that two parties completed a protocol execution with the expected outcome. We can then use such reasoning to define authentication, as illustrated further in this section.

Second, *phases* introduce a temporal ordering that lets us control when the adversary gains access to specific secrets. Analysis starts at phase 0, and each subsequent phase can disclose additional material. For example:

```
phase 1; out(c, seed_x)
```

This declares the start of phase 1 and sends the seed value `seed_x` on the public channel `c`, making it available to the attacker. By querying at different phases, we can observe the consequences of leaking certain values. For instance, we can model the compromise of long-term key material at a later phase and check whether the attacker is able to recover messages sent during phase 0, prior to the leakage. This is one way to reason about forward secrecy.

8.2. Formal Security Properties

We evaluate the security goals from Sect. 3.1 using *ProVerif* queries. In this subsection, we illustrate the use of three main query types with examples from our CASE model.

Confidentiality using Secrecy Queries. We test whether the adversary can derive a secret value using *attacker queries*. The following example targets the responder's session keys:

```
1 not attacker(q_session_keys_r) phase 2.
2 query attacker(q_session_keys_r).
```

Line 1 asks whether the session key remains secret through the end of phase 2 (before certain material is leaked). Line 2 asks whether the adversary can ultimately derive it after all phases complete. If the first query returns *true* (meaning the key is still secret at phase 2) and the second returns *false* (meaning the adversary can derive it after all phases), we can conclude that a value leaked after phase 2 enabled the attack. If that leaked value is a long-term secret, this constitutes a violation of forward secrecy.

Mutual Authentication using Correspondence Queries. To verify authentication, we use queries that reason about events emitted during protocol execution, such as receiving a message, verifying a signature, or completing a protocol run. To prove mutual authentication, we check that whenever an initiator believes it has completed a protocol execution with certain parameters, an honest responder started and completed the same execution with matching parameters, and vice versa. In *ProVerif*:

```
query noc: group, k: group, r: bitstring;
  inj-event(initiator_end(noc, k, r))
  ==> inj-event(start_session_with_initiator(k, r)
    ).
```

This reads as: whenever the initiator completes a session (recording `initiator_end` with identity `noc`, key material `k`, and randomness `r`), a responder must have previously accepted a session with the same `k` and `r`. The `inj` prefix (injective) further requires that each initiator completion maps to a *distinct* responder session, ruling out replay attacks. This query checks one direction (responder authenticates to initiator). To achieve mutual authentication, we add the symmetric query for the other direction and require both to hold.

Resistance to Offline Guessing using Observational Equivalence Queries. Password-based protocols require a third kind of query. Declaring a value a *weaksecret* (`weaksecret pw`) asks whether an adversary can verify a guess at it offline, without interacting with an honest party. If this holds, no offline dictionary attack on the passcode exists in the model.

8.3. Verification Results

We present our results in this subsection and invite readers to inspect our documented models and reproduce the results for a more detailed understanding.

8.3.1. PASE

Our models of PASE and SPAKE2+ yield matching security properties. We present the findings for PASE below.

Authentication. We establish authentication through injective correspondences. Whenever a party completes PASE, the model requires that every earlier step of the run occurred in order on both sides (with matching context and state), and that the prover started with the real passcode:

```
inj-event(verifier_success(ctx,X,Y,k)) ==>
  inj-event(prover_send_pake3(ctx,X,Y,k)) && ...
  && inj-event(prover_uses_passcode(ctx,pw,w0))
  && inj-event(verifier_uses_credential(ctx,w0,L))
    is true.
```

The symmetric query for the prover also holds, as do the pairwise correspondences chaining each step to its predecessor in the protocol's order. Together these queries establish agreement on the full transcript: each completion is backed by a distinct peer session, ruling out replays. This remains

a weak form of authentication: PASE only verifies passcode knowledge, not the identity of the peer. The prover only learns that its peer holds the matching verifier, not that it is the intended device; the verifier only learns that its peer holds the corresponding passcode. Since no long-term identity keys are used, no stronger notion of authentication can be achieved. This enables the Controller-in-the-Middle attack described in Sect. 7.2.

Confidentiality. The session keys derived during PASE remain secret from the adversary:

```
Query not attacker(q_confirm_p[]) is true.
Query not attacker(q_confirm_v[]) is true.
Query not attacker(q_i2rkey[]) is true.
Query not attacker(q_r2ikey[]) is true.
```

All four queries return *true*, confirming that neither the confirmation keys nor the initiator-to-responder and responder-to-initiator session keys can be derived by the adversary.

Resistance to offline guessing. PASE resists offline dictionary attacks on the passcode (`Weak secret pw is true`). An adversary who observes the messages sent on the wire, even across protocol executions, cannot test passcode guesses offline. This changes once the verifier material leaks: in the variant that discloses w_0, the same query returns *false*, which confirms formally that w_0 alone enables the offline search of Sect. 7.1.

Forward secrecy. We test whether previously established session keys remain secret even if the passcode is later compromised. To do so, we leak it in phase 1 via `(!(phase 1; out(c, pw)))`. All four key queries from the confidentiality analysis continue to hold, meaning that an attacker who obtains the passcode after a session has been established cannot decrypt that earlier session's traffic.

8.3.2. CASE

Under the default *ProVerif* adversary (full network control, no compromised secrets), CASE satisfies all of Matter's claimed security properties.

Mutual authentication. Both injective correspondence queries return *true*, confirming that each party's session completion is backed by a distinct, genuine session from the peer.

Confidentiality. *ProVerif* confirms that the session keys derived during CASE remain secret from the adversary. Neither the initiator-to-responder nor the responder-to-initiator session key can be derived, even by an adversary with full control of the network.

Identity Protection. CASE provides identity protection for both initiator and responder. This is achieved through the Identity Protection Key (IPK), a symmetric key shared among all members of a fabric that is used to encrypt identity information during the CASE handshake. As long as the adversary does not possess the IPK, it cannot learn which parties are communicating.

Stretched adversary analysis. To understand CASE's resilience to state compromise, we progressively leak secret values to the adversary. We leak the initiator's ephemeral key in phase 1, the responder's ephemeral key in phase 2, and the IPK in phase 3:

```
(! (phase 1; out(c, seed_x))) |
(! (phase 2; out(c, seed_y))) |
(* Attacker joins the Fabric and obtains IPK *)
(! (phase 3; out(c, ipk)))
```

With all three values leaked (after phase 3), the adversary can derive identity keys, shared secrets (the raw Diffie-Hellman values), and session keys (derived from the shared secrets via a key derivation function):

```
(* Identity Keys *)
Query not attacker_p3(q_id_i_i[]) is false.
Query not attacker_p3(q_id_i_r[]) is false.
(* Shared Secrets *)
Query not attacker_p3(q_shared_secret_i[]) is false.
Query not attacker_p3(q_shared_secret_r[]) is false.
(* Session Keys *)
Query not attacker_p3(q_session_keys_i[]) is false.
Query not attacker_p3(q_session_keys_r[]) is false.
```

However, if the IPK is not leaked (i.e., the adversary only has both ephemeral keys but is not a fabric member), the picture changes. Identity keys and session keys remain protected. The shared secrets (raw Diffie-Hellman values) are derivable since the adversary has both ephemeral keys, but the session keys cannot be computed without the IPK, which is an additional input to the key derivation function:

```
Query not attacker_p2(q_id_i_i[]) is true.
Query not attacker_p2(q_id_i_r[]) is true.
Query not attacker_p2(q_shared_secret_i[]) is false.
Query not attacker_p2(q_shared_secret_r[]) is false.
Query not attacker_p2(q_session_keys_i[]) is true.
Query not attacker_p2(q_session_keys_r[]) is true.
```

This identifies the IPK as a critical trust anchor for CASE's key derivation: for the adversary to compromise session keys or peer identities, it must obtain both an ephemeral key *and* gain access to the IPK. Obtaining the IPK requires either being a member of the same fabric or receiving it from a fabric insider. The IPK, however, is a symmetric key shared by every node of a fabric, so its secrecy rests on that of the weakest fabric member.

8.3.3. CASE Resumption

As described in Sect. 5.2, CASE Resumption allows efficient derivation of new session keys from previously existing key material without execution of elliptic-curve arithmetic or signature verification.

Security degradation. Our analysis reveals that a resumed session is significantly less secure than a standard CASE session. Leaking a *single* ephemeral private key from the preceding CASE session, without the IPK, already compromises the resumed session keys. Leaking only the initiator's ephemeral key gives:

```
Query not attacker_p1(q_eph_x[]) is false.
Query not attacker_p1(q_eph_y[]) is true.
Query not attacker_p1(q_shared_secret_i[]) is false.
(* Session keys of the full handshake remain protected *)
Query not attacker_p1(q_session_keys_i[]) is true.
(* Session keys from a resumed session are compromised *)
Query not attacker_p1(q_resumption_session_keys_i[]) is
     false.
Query not attacker_p1(q_resumption_session_keys_r[]) is
     false.
```

The responder's ephemeral private key remains secret, and the session keys of the full handshake are still protected by the IPK (*true*), yet the shared secret and with it the resumed session keys are derivable (*false*). Leaking the responder's ephemeral instead yields the same outcome, so either one suffices. The IPK no longer acts as a protective layer around session keys: unlike in standard CASE, where the adversary needs both an ephemeral private key and the IPK, here one ephemeral key alone is enough. The attacker no longer needs to be a member of the fabric or obtain the IPK from a fabric insider. This is particularly concerning because, in practice, CASE Resumption is likely the most frequently executed protocol by Matter devices.

8.4. Patching CASE Resumption

As a fix, we propose that both nodes derive, at the end of the CASE handshake, a dedicated resumption secret *ResumptionPreKey* = KDF(*SharedSecret*, *IPK* ‖ Hash(Msg1 ‖ Msg2 ‖ Msg3), "ResumptionPreKey") and store it instead of the *SharedSecret*, which can then be erased together with the ephemeral keys. S1RK, S2RK and the resumption session keys would then be derived from *ResumptionPreKey* instead, and the messages on the wire remain unchanged. We verify this patched design in a separate model in our artefact, and confirm that the same results for CASE hold.

8.5. Verdict

Our formal analysis confirms that, under the symbolic model, Matter's key establishment protocols satisfy the security properties targeted by their designers, with the exception of the resumption protocol. PASE provides confidentiality and forward secrecy, and resistance to offline guessing, though only a weak form of authentication based on passcode knowledge. CASE provides mutual authentication, confidentiality, and identity protection.

The main concern raised by our analysis is related to the CASE Resumption protocol, which degrades security compared to the full CASE handshake. Our analysis shows that the secrecy of resumed session keys rests solely on the secrecy of the ephemeral keys used in CASE. Since resumption is the protocol used for every session after initial commissioning, leakage of a single ephemeral key compromises the majority of a device's sessions. As shown above, however, this weakness can be easily patched without changing the messages on the wire.

8.6. Limitations

As with any formal modelling effort, our security guarantees hold exclusively within the boundaries of our model. We now outline the key limitations.

Symbolic vs. computational model. Our analysis operates in the symbolic model, where cryptographic primitives are assumed to be ideal. In particular, SPAKE2+ and PASE rely on elliptic curve group arithmetic, and our symbolic modelling of these operations does not provide guarantees as strong as a proof in the computational model. ProVerif does not support algebraic manipulation, and is limited to the equational theories and reductions defined in our model. Where a property fails, ProVerif returns an attack trace, which lists the steps the attacker takes and can serve as the starting point for a reduction.

Scope of the model. We analyse PASE, CASE, and CASE Resumption independently without connecting them into the full commissioning flow of Fig. 1. We do not model the DAC verification component and assume that it is secure.

Security definitions. Matter does not define its security goals using standard formal definitions. Our security claims are therefore bound to the definitions, simplifications, and assumptions we describe in this paper.

Entropy assumption. Our model assumes that all random values are generated with sufficient entropy (Sect. 3.2). If this assumption does not hold, the security guarantees of the analysed protocols are compromised, as we demonstrate for CASE Resumption. We leave measuring the entropy actually available on deployed devices as open work.

9. Discussion and Mitigations

Table 1 summarises our findings: the capability an attacker needs to exploit each of them, their consequences, the affected Matter threats and countermeasures, the supporting evidence, and their disclosure status. In this section, we discuss concrete mitigations for these weaknesses. Several of our recommendations were shared with the Matter team through a responsible disclosure process, as discussed in Appendix A.

Passcode entropy, not fresh salts. A natural response to our passcode-recovery attack (Sect. 7.1) is to derive a fresh salt per session. While this would stop an attacker from reusing one table across sessions, it would not prevent recovery: once w_0 and the matching salt are known, the passcode space can still be exhausted offline. Moreover, since a new salt changes (w_0, L), the device would have to keep the passcode and recompute the verifier each session, so that a compromised device now leaks the passcode. The root cause is the entropy of the passcode, not the salt. To raise it, we propose switching the 11-digit numeric manual pairing code to a case-insensitive alphanumeric format of the same length, which would yield approximately 57 bits with minimal usability impact. In our discussions, Matter attributed the numeric-only design to usability across regions, particularly locales that do not use Latin alphabets.

Hardening PASE with an oblivious PRF. The attack of Sect. 7.1 relies on two ingredients: a passcode space small enough to enumerate (10^8), and a way to evaluate $pw \mapsto w_0$ without the device, which the public salt provides. The first cannot change within Matter's usability constraints, so we remove the second. Following the standard construction that places an *oblivious PRF* (OPRF) in front of an augmented PAKE [28], we replace the salt by a per-device secret key k, applied to the passcode without ever leaving the device. Unlike a fresh salt, which the commissioner must receive and is therefore public, k is used obliviously: the commissioner learns $F_k(pw)$ but nothing about k, and the device, or anyone impersonating it, learns nothing about pw. The

TABLE 1. SUMMARY OF FINDINGS: REQUIRED ATTACKER CAPABILITY, CONSEQUENCE, AFFECTED MATTER THREAT (T) OR COUNTERMEASURE (CM), EVIDENCE, AND STATUS.

Finding	Attacker needs	Leads to	Matter T/CM	Evidence	Status
Specification					
Low-entropy passcode	Leaked w_0	Passcode recovery	CM5, T102	Spec.	Disclosed
Weak PBKDF2 parameters	Leaked w_0	Passcode recovery	CM99, T102	Spec., devices	Disclosed
No group-membership check	–	None known	–	Spec.	Disclosed
Commissioner not authenticated	Passcode	CitM	T173–T175	Spec.	Disclosed
Attestation not enforced	Device impersonation	CitM	–	Spec.	Disclosed
CASE Resumption downgrade	Leaked ephemeral key	Resumed keys	–	ProVerif	Disclosed
Implementation					
Retry limit not enforced	Network access	Online guessing	CM100, T101, T112	Code	Patched
Window not enforced	Network access	Online guessing	–	Code	Patched
Exclusivity not enforced	Passcode	Session takeover	CM3, T15	Code	Patched
No group-membership check	–	None known	–	Code	Patched
Attacks					
Passcode recovery	Leaked w_0	Device control	T102	PoC, devices	Disclosed
Controller-in-the-Middle (CitM)	Passcode	Full MitM	T15, T173–T175	Spec.	Disclosed

device is provisioned with (k, w_0, L) instead of $(salt, w_0, L)$, where $w_0 \parallel w_1 = \text{PBKDF}(F_k(pw))$. The OPRF rides on the existing PBKDFParam exchange: the commissioner draws a random scalar r and sends $B = r \cdot H_G(pw)$, where H_G hashes the passcode to a group element. The device checks that B is a valid point and returns $k \cdot B$. The commissioner multiplies $k \cdot B$ by r^{-1} to obtain $k \cdot H_G(pw)$, computes $F_k(pw) = \text{Hash}(pw \parallel k \cdot H_G(pw))$, and derives w_0, w_1 as before. SPAKE2+ then runs unchanged, and the PBKDF runs on the commissioner only. This extension is intended to protect against offline passcode recovery from leaked verifier values (w_0, L), provided that k remains secret. We leave the evaluation of this design, including its security and its costs on constrained devices, to future work.

Hardening CASE Resumption. Aside from the patch proposed in Sect. 8.4, CASE Resumption could be hardened further at the cost of additional Diffie-Hellman operations. For instance, each party could generate a fresh ephemeral keypair and feed the resulting shared secret into the resumption key derivation alongside the stored secret, so that session keys can no longer be derived from stored state alone. This remains cheaper than full CASE and negligible for mains-powered or frequently recharged devices.

Mandating group membership checks. SPAKE2+ requires each party to verify that a received public key is a valid group element, but the Matter specification never states this requirement. We find no exploit today (Sect. 4.2), yet that safety is incidental to Matter's current parameters, so the check should still be mandated. The Alliance is informed and is investigating this specification-level gap.

Authenticating the commissioner. As demonstrated by the CitM attack (Sect. 7.2), PASE does not authenticate the commissioner beyond verifying knowledge of the passcode. Any entity that knows the passcode can commission the device. We note this as a fundamental design constraint of Matter's commissioning model that warrants further investigation. A good starting point is the work of Shashwat et al. [15], in which they propose a two-way attestation scheme in which the device also verifies the commissioner's certificate before proceeding with commissioning.

We notice a pattern in Matter's design with a tendency to mandate a single cryptographic suite for all devices, sized to run on the most resource-constrained hardware. As a result, the entire ecosystem is held to the security level of its weakest members, and capable devices cannot take advantage of stronger primitives. We advocate for a more crypto-agile approach that maximises security. More broadly, secure parameters such as minimum iteration counts, as well as checks such as retry limits, should be enforced at the specification level and verified during certification. As our implementation assessment shows, even security-critical requirements can slip through in certified codebases.

10. Conclusion

Throughout this study, we sought to evaluate Matter against the high standards it set through its ambitious goals as a secure and interoperable IoT protocol. As such, we presented a protocol-level analysis of its core key establishment mechanisms, focusing on PASE, CASE, and CASE Resumption. At the specification level, we identified weaknesses that are not visible from the underlying SPAKE2+ and SIGMA protocols alone. These include low-entropy setup passcodes, weak PBKDF parameters, and long-lived device security materials. At the implementation level, we found that certified code failed to enforce security-critical protections. All these vulnerabilities were responsibly disclosed and have since been patched in the official codebase. We validated these findings through two attacks: a passcode recovery attack, and a controller-in-the-middle (CitM) attack during recommissioning. Both are feasible under Matter's own threat model. Our formal verification confirmed that PASE and CASE satisfy their claimed security properties in the

symbolic model, but also revealed that the CASE Resumption protocol degrades security compared to the full CASE handshake. Based on these results, we showed that several countermeasures in Matter's threat model are insufficient in practice and proposed specification-level improvements. We hope that this analysis provides a foundation for future work on Matter and other cross-vendor IoT standards (e.g., evaluating the CitM on commercial devices), helping guide their evolution towards stronger security guarantees.

Acknowledgements

We thank the members of the Connectivity Standards Alliance for their time, responsiveness, feedback and contributions throughout the disclosure procedure as well as for their review of this document. We also thank Karthikeyan Bhargavan for his insightful feedback related to this project. Arman further thanks Carmela Troncoso for her guidance throughout his internship at MPI-SP.

This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02, CyberSecurity Research Flanders with reference number VR20192203, and by the European Union's Horizon Research and Innovation program under grant agreement No. 101119747 (TELEMETRY). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Cybersecurity Research Flanders or the European Union.

References

[1] *Matter Core Specification Version 1.5.1*, Connectivity Standards Alliance (CSA), 2026, available online. [Online]. Available: https://csa-iot.org/developer-resource/specifications-download-request/

[2] P. DeLoatch, "Matter's mission to unify smart homes from apple, amazon, and google," *The Verge*, 2021. [Online]. Available: https://www.theverge.com/22787729/matter-smart-home-standard-apple-amazon-google

[3] P. Monckton, "Matter smart home standard explained: Here's why google, apple, and amazon are backing it," *Tom's Guide*, 2022. [Online]. Available: https://s.42l.fr/Matter

[4] *Matter Security and Privacy Fundamentals*, Connectivity Standards Alliance (CSA), 2022, available online. [Online]. Available: https://csa-iot.org/wp-content/uploads/2022/03/Matter_Security_and_Privacy_WP_March-2022.pdf

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[6] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," in *2019 IEEE Symposium on Security and Privacy (S&P)*, 2019.

[7] S. Duttagupta, S. Wyns, N. Antonijević, D. Singelée, and B. Preneel, "One Tap to Hijack Them All: A Security Analysis of the Google Fast Pair Protocol," in *2026 IEEE Symposium on Security and Privacy (S&P)*, 2026.

[8] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," in *2017 IEEE Symposium on Security and Privacy (S&P)*, 2017.

[9] E. Ronen and A. Shamir, "Extended Functionality Attacks on IoT Devices: The Case of Smart Lights," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[10] T. Taubert and C. A. Wood, "SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol," RFC 9383, Sep. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9383

[11] H. Krawczyk, "Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike protocols," in *Advances in Cryptology - CRYPTO 2003*, 2003, pp. 400–425.

[12] Connectivity Standards Alliance (CSA), "Matter-certified products listing," number retrieved on 25/05/2026. [Online]. Available: https://csa-iot.org/csa-iot_products/

[13] V. Shoup, "Security analysis of SPAKE2+," Cryptology ePrint Archive, Paper 2020/313, 2020.

[14] M. Loos, "Security analysis of the matter protocol," *Master Thesis*, 2023.

[15] K. Shashwat, F. Hahn, X. Ou, and A. Singhal, "Security analysis of trust on the controller in the matter protocol specification," in *2023 IEEE Conference on Communications and Network Security (CNS)*, 2023, pp. 1–6.

[16] S. Liao, J. Yan, and L. Cheng, "WIP: Hidden Hub Eavesdropping Attack in Matter-enabled Smart Home Systems," in *Workshop on Security and Privacy in Standardized IoT (SDIoTSec)*, 2024.

[17] N. Shafqat and A. Ranganathan, "Seamlessly Insecure: Uncovering Outsider Access Risks in AiDot-Controlled Matter Devices," in *2024 IEEE Security and Privacy Workshops (SPW)*, 2024.

[18] H. Wang, Y. Fang, Y. Liu, Z. Jin, E. Delph, X. Du, Q. Liu, and L. Xing, "Hidden and lost control: on security design risks in iot user-facing matter controller," in *In Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.

[19] F. Zohdi and D. Antonioli, "MaDoS: Matter DoS Attacks via Secure Channel Status Reports," in *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2026.

[20] K. Bhargavan, C. Jacomme, F. Kiefer, and R. Schmidt, "Formal verification of the PQXDH Post-Quantum key agreement protocol for end-to-end secure messaging," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[21] [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2

[22] T. Taubert and C. A. Wood, "SPAKE2+, an Augmented PAKE," Internet Engineering Task Force, Internet-Draft draft-bar-cfrg-spake2plus-02, Dec. 2020.

[23] T. Jager, J. Schwenk, and J. Somorovsky, "Practical Invalid Curve Attacks on TLS-ECDH," in *ESORICS 2015*, 2015.

[24] A. Antipa, D. Brown, A. Menezes, R. Struik, and S. Vanstone, "Validation of Elliptic Curve Public Keys," in *PKC*, 2003.

[25] C. H. Lim and P. J. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup," in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed., 1997.

[26] C. S. Alliance, "Working with the chip tool: attestation-related flags," 2026. [Online]. Available: https://github.com/project-chip/connectedhomeip/blob/master/docs/development_controllers/chip-tool/chip_tool_guide.md#attestation-related-flags

[27] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.05: Automatic cryptographic protocol verifier, user manual and tutorial," https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf, 2023.

[28] S. Jarecki, H. Krawczyk, and J. Xu, "Opaque: An asymmetric pake protocol secure against pre-computation attacks," in *Advances in Cryptology – EUROCRYPT 2018*, 2018, pp. 456–486.

[29] C. S. Alliance, "Matter," 2025. [Online]. Available: https://github.com/project-chip/connectedhomeip

[30] M. Dworkin, "Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality," National Institute of Standards and Technology, Special Publication 800-38C, 2004.

[31] E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators," National Institute of Standards and Technology, Special Publication 800-90A Revision 1, 2015.

[32] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," National Institute of Standards and Technology, Special Publication 800-90B, 2018.

[33] W. C. B. Elaine Barker, "Recommendation for password-based key derivation," National Institute of Standards and Technology, Special Publication 800-132, 2010.

[34] L. Chen, D. Moody, R. Peralta, R. Sudharshan, and A. Regenscheid, "Recommendation for discrete logarithm-based cryptography: Elliptic curve domain parameters," National Institute of Standards and Technology, Special Publication (Draft) 800-186, 2019.

Appendix A. Ethics Considerations

We adhered to responsible disclosure practices in line with established norms in security research. On 11 October 2024, we contacted the Connectivity Standards Alliance, the governing body behind Matter, to disclose our findings related to multiple security vulnerabilities in the Matter protocol. The Alliance formally acknowledged our report on 15 October 2024, and we agreed on a 150-day disclosure window to provide time for internal review and remediation.

Over the following months, we engaged in several rounds of discussions with the Alliance, including detailed email exchanges and online meetings with their engineering, security and top management teams. During these sessions, we provided full technical justifications for our findings and outlined potential cryptographic improvements. Specifically, we recommended replacing PBKDF2 with memory-hard alternatives such as Argon2, enforcing unique salt generation per session, and increasing the entropy of commissioning passcodes. The Alliance confirmed the feasibility of several proposals, including the move to alphanumeric codes for passcodes and the potential adoption of stronger key derivation functions.

In addition to our protocol-level analysis, we identified critical implementation vulnerabilities in the certified open-source JavaScript Matter SDK (*matter.js*). The issues were promptly acknowledged, patched, and merged into the main codebase. At the specification level, the Alliance is still investigating the missing group-membership check.

Importantly, our research is based entirely on the Matter Core Specification documents [1] (versions 1.3, 1.4, and 1.5.1), a comprehensive ~1200-page draft, and the official Matter SDK [29], both of which are publicly available. We did not reverse-engineer any proprietary firmware, nor did we target commercial devices in ways that would violate legal or ethical boundaries.

Our intent throughout this project has been to identify and responsibly report weaknesses that could be exploited in practice, with the ultimate goal of strengthening the security posture of Matter as a widely deployed IoT standard.

Appendix B. Cryptographic Primitives in Matter

Matter secures communication, authentication, and data integrity with a fixed set of cryptographic primitives, largely following NIST guidelines [30], [31], [32], [33], [34]. Randomness comes from deterministic random bit generators (CTR, Hash, and HMAC DRBGs, built on AES and SHA-256/SHA-512), complemented by a true random number generator that provides non-deterministic entropy for key and nonce generation where the hardware supports it. SHA-256 is the primary hash function, and HMAC-SHA-256 provides message authentication. Public-key operations use the NIST P-256 curve (secp256r1): ECDSA for digital signatures and ECDH for key agreement. Symmetric encryption uses AES-CCM authenticated encryption, protecting the confidentiality, integrity, and authenticity of data in transit. Keys are derived with HKDF based on HMAC-SHA-256, while PBKDF2 (HMAC-SHA-256 with a public salt and configurable iteration count) derives keys from passcodes, mainly during commissioning.

Appendix C. Node Operational Certificate (NOC)

The Node Operational Certificate (NOC) is a core component of Matter's identity and access control infrastructure. Each commissioned device receives a unique NOC, which enables secure interactions within the fabric by providing a cryptographically authenticated device identity.

The commissioner obtains the NOC after successful device attestation. Key steps in the NOC issuance include:

1) After DAC validation, the device generates a keypair and submits a Certificate Signing Request (CSR) to the commissioner requesting a NOC.
2) The commissioner relays this request to the fabric's trusted CA.
3) If it is valid, the CA generates and signs a NOC, so that the device's cryptographic identity is tied to the fabric.
4) The NOC is relayed back to the device by the commissioner, where it is securely stored, providing the basis for mutual authentication with other devices.

Appendix D. Shared Key Derivation Differences

In the SPAKE2+ RFC document, the key derivation process is as follows:

```
  (1) K_main = Hash(TT)
  (2) K_shared = KDF(nil, K_main,
"SharedKey")
  (3) K_confirmP ‖ K_confirmV =
       KDF(nil, K_main,
"ConfirmationKey")
```

Here a key K_{main} is derived from the hash of the transcript TT as an intermediate step. Using K_{main}, we make two distinct KDF calls to derive the shared secret K_{shared}, and the confirmation keys $K_{confirmP}, K_{confirmV}$.

The key difference from the second draft version is that the shared key K_e is one half of the hash of the transcript TT. The method from the final SPAKE2+ specification is thus potentially more secure as it passes the entire hash into a key derivation function to create the shared value. This might be helpful in completely decoupling the shared secret from the protocol transcript.

Appendix E.
PBKDF2 Pre-Computation Benchmark Results

We benchmarked PBKDF2-HMAC-SHA256, which Matter uses to derive the pair (w_0, w_1) in PASE. Measurements were taken on a MacBook Air with an Apple M2 chip, 16 GB RAM, and macOS Sequoia 15.3.2. Table 2 reports times for one derivation and for exhausting the valid 8-digit passcode space.

TABLE 2. PBKDF2-HMAC-SHA256 DERIVATION COST AND ESTIMATED TIME TO PRE-COMPUTE A LOOKUP TABLE OVER THE FULL 8-DIGIT PASSCODE SPACE. T_1 IS THE TIME ON A SINGLE CPU CORE, WHILE T_{128} ASSUMES IDEAL LINEAR SCALING ACROSS 128 CORES.

Iterations	Per derivation	T_1	T_{128}
1 000	361.34 μs	10 h	5 min
10 000	3.60 ms	100 h	47 min
100 000	35.95 ms	42 d	8 h
600 000	214.24 ms	248 d	47 h
1 000 000	353.72 ms	409 d	3 d

The minimum iteration count allowed by Matter (1,000) yields a full lookup table in about ten hours on a single core, or just five minutes on 128 cores. Even the maximum permitted count (100,000) can be pre-computed in eight hours on a multi-core setup.

While generating the table may require significant time, once built, the lookups are fast. We measured the time required to look up a w_0 value in the pre-computed lookup table. Using a naive linear search implementation, the median lookup time was approximately 33 s. This represents a worst-case scenario without any optimisations. With standard techniques such as binary search or hash maps, and on a more powerful CPU, lookup times could be reduced to the order of milliseconds.

Appendix F.

F.1. SPAKE2+

SPAKE2+ is an augmented PAKE protocol designed for secure password-based authentication [10]. It enables two parties, the *prover* and the *verifier*, to establish a strong shared secret based on a password without revealing the password itself. The augmented nature of SPAKE2+ means that only one party, the prover, directly uses the password. The verifier, on the other hand, only has access to a record derived from the password, created during an initial offline registration phase. Since Matter's implementation is based on the second draft of this protocol, we discuss the protocol design as specified in that draft.

The protocol relies on several key elements:

Group G. Cyclic group in which the Computational Diffie-Hellman (CDH) problem is hard. G has order $p \cdot h$, where p is a large prime and h is the cofactor.

Generator P. Generator of the subgroup of order p, used to generate group elements during the key exchange.

Identities. The identities of the prover, A, and the verifier, B, are used in the key derivation process to bind the derived keys to the participants.

SPAKE2+ consists of two main phases: an offline registration phase and an online authentication phase. During the offline phase, the prover derives, from the password and the identities, an integer value via a PBKDF**. This integer is then split into two halves, w_{0s} and w_{1s}, which are each reduced modulo p, resulting in values w_0 and w_1.

The prover shares w_0 and $L = w_1 \cdot P$ with the verifier. It is important to note that the verifier does not possess, and therefore does not store, the value w_1 in its "registration record." The prover computes,

$$w_{0s} \parallel w_{1s} = \text{PBKDF}(\text{pw} \parallel A \parallel B, \text{salt}, \text{iteration_count})$$
$$w_0 = w_{0s} \bmod p$$
$$w_1 = w_{1s} \bmod p$$

Next, in the online phase, two public group elements M and N of unknown discrete logarithm are generated from the prime-order subgroup of G. These elements may be pre-computed and re-used. The specification of SPAKE2+ suggests pre-computed values, which Matter adopts.

Messages are then exchanged to compute the shared secret, as illustrated in Fig. 5. The steps are detailed below:

1) The prover initiates the protocol by:
   a) Generating a random integer x from $[0, p-1]$.
   b) Computing the ephemeral public value pA, with $pA = x \cdot P + w_0 \cdot M$.
   c) Sending pA to the verifier.
2) The verifier, upon receiving pA, performs the following:
   a) Checks if pA is a member of the large prime-order subgroup of G. If not, the verifier aborts.
   b) Generates a random integer y from $[0, p-1]$.
   c) Computes the ephemeral public share pB, with $pB = y \cdot P + w_0 \cdot N$.
   d) Sends pB to the prover.
   e) Computes the common values:
   $Z = h \cdot y \cdot (pA - w_0 \cdot M)$
   $V = h \cdot y \cdot L$.
3) The prover, upon receiving pB, does the following:
   a) Checks pB for group membership.
   b) Computes the common values:
   $Z = h \cdot x \cdot (pB - w_0 \cdot N)$
   $V = h \cdot w_1 \cdot (pB - w_0 \cdot N)$.

**In reality, the length of each item is also incorporated into the PBKDF input, but we have omitted these details to simplify the explanation.

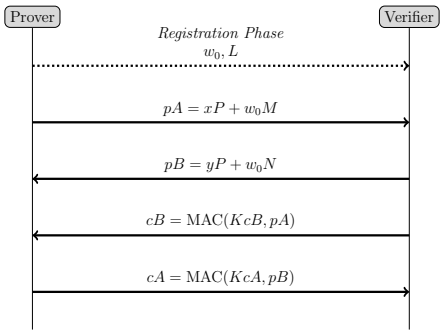

Figure 5. SPAKE2+ protocol description. The offline registration phase, during which the necessary common values (such as w_0 and L) are generated and shared, is represented with dotted lines.

After public value exchange and computation of the shared values, both parties derive a shared secret *Ke* from the protocol transcript *TT*, defined as follows[††]:

$$\begin{aligned} TT = {} & \mathrm{len}(A) \parallel A \parallel \mathrm{len}(B) \parallel B \parallel \mathrm{len}(M) \parallel M \\ & \parallel \mathrm{len}(N) \parallel N \parallel \mathrm{len}(pA) \parallel pA \parallel \mathrm{len}(pB) \parallel pB \\ & \parallel \mathrm{len}(Z) \parallel Z \parallel \mathrm{len}(V) \parallel V \parallel \mathrm{len}(w_0) \parallel w_0 \end{aligned}$$

The secrets *Ka* and *Ke* are derived as follows:
$Ka \parallel Ke = \mathrm{Hash}(TT)$

Each key is half the length of the hash output. From *Ka*, confirmation keys *KcA* and *KcB* are derived using a KDF:
$KcA \parallel KcB = \mathrm{KDF}(\mathrm{nil}, Ka, \text{“ConfirmationKeys”})$

The confirmation keys are used to generate and verify key confirmation messages, ensuring both parties agree on the derived secrets before proceeding. Specifically, the prover sends $cA = \mathrm{MAC}(KcA, pB)$ and the verifier sends $cB = \mathrm{MAC}(KcB, pA)$[‡‡]. Both sides then check if the expected MAC value matches the received one. This ensures both parties derive the same secret, as the MAC binds confirmation to the exchanged ephemeral values (pA, pB).

Once keys are confirmed, *Ke* becomes the authenticated shared secret for all future encryption. It is important to note that *Ka* and its derived keys serve no purpose beyond key confirmation and should be securely erased from memory.

F.2. SIGMA Protocols

The SIGMA family of key-exchange protocols is designed to provide certificate-based security [11]. It supports scenarios where identity protection may or may not be required. SIGMA protocols achieve this using a combination of Diffie-Hellman exchanges authenticated by digital signatures and MACs. Several variants exist, each suited to different threat models. The basic SIGMA protocol does not offer identity protection. The SIGMA-I and SIGMA-R variants both protect each party's identity against passive eavesdroppers, but differ in which party is additionally protected against active attackers: SIGMA-I protects the initiator's identity, while SIGMA-R protects the responder's.

[††]The SPAKE2+ specification incorporates a *Context* component (such as the application name or version) into the calculation of TT. We omit this detail for clarity.

[‡‡]In the SPAKE2+ specification, a fixed string may be used instead of pB and pA.

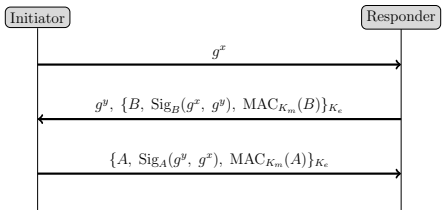

Figure 6. SIGMA-I protocol description. The initiator here is the commissioner, whose identity is protected from active attackers.

We focus on the SIGMA-I variant, which is particularly well-suited for scenarios requiring protection of the initiator's identity against active attackers, and the responder's identity from passive attackers. For example, when a client (initiator) connects to a server (responder), it may be more important to conceal the user's identity than that of the server.

Fig. 6 illustrates the protocol flow of SIGMA-I:

- g^x and g^y are the Diffie-Hellman public values.
- $\mathrm{Sig}_B(g^x, g^y)$ and $\mathrm{Sig}_A(g^y, g^x)$ are digital signatures on the exchanged values.
- $\mathrm{MAC}_{K_m}(\mathrm{B})$ and $\mathrm{MAC}_{K_m}(\mathrm{A})$ are the MACs computed over the sender's identity using a MAC key K_m derived from g^{xy}.
- $\{...\}_{K_e}$ denotes encryption using a key K_e, also derived from g^{xy}.

The main components of the protocol are the following:

Diffie-Hellman Exchange. Each party generates an ephemeral Diffie-Hellman key pair and exchanges the public values, g^x and g^y. This ensures forward secrecy.

Signatures and MACs. Both parties sign the exchanged Diffie-Hellman values to ensure integrity and authenticity. Including the peer's DH value in the signed message also prevents replay attacks [11]. MACs are computed over each party's identity using a key derived from the shared secret g^{xy}. These MACs serve as "proof of possession" of the session key and bind each identity to the key, preventing identity misbinding attacks – where an attacker relays messages between honest parties to make them believe they are talking to each other.

Encryption for Identity Protection. To provide identity protection, the identities and signatures are encrypted using a key derived from the shared Diffie-Hellman secret g^{xy}.

F.3. Differences between SPAKE2+ and PASE

This subsection outlines the differences we have identified between the official SPAKE2+ specification and Matter's PASE implementation. As noted in Sect. 4, Matter is based on the second draft of SPAKE2+ [22], which differs from the final RFC in key confirmation and derivation (see Appendix D). Beyond these draft-level differences, PASE introduces the following changes:

PBKDF Parameters Sharing. Matter introduces additional messages to exchange PBKDF parameters, so that both parties agree on the same key-derivation inputs.

Session ID and Random Number. Each message in PASE includes a session ID and a random number. The session ID uniquely identifies the session, while the random number provides resistance to replay attacks.

Lack of Group Membership Checks. As discussed in Sect. 4.2, Matter does not perform the group membership checks required by the SPAKE2+ specification.

Final Keys. From the shared secret *Ke*, Matter derives two session keys: *I2RKey*, used by the initiator to encrypt and authenticate messages (and by the responder to decrypt and verify them), and *R2IKey* for the reverse direction.

F.4. Differences between SIGMA-I and CASE

CASE adapts the SIGMA-I design with the following changes:

AEAD instead of separate MAC. SIGMA uses a separate MAC alongside encryption to bind each party's identity to the session, with distinct keys for MACing and encrypting. Matter replaces this with a single AEAD operation: each party signs the ephemeral public keys and its certificate, then encrypts both under AEAD. The AEAD tag implicitly provides the identity binding of SIGMA's separate MAC.

Resumption ID. The responder sends a resumption ID, allowing later resumption without a full handshake.

Destination identifier. The initial message includes a destination identifier, which tells the responder which fabric the initiator has selected.

Random values and session IDs. Unlike SIGMA, each CASE party generates a random value (*initiatorRandom*, *responderRandom*) to ensure session uniqueness, and a session ID (*initiatorSessionId*, *responderSessionId*) for session management.

Per-message encryption keys. Unlike SIGMA, CASE derives S_{2K} and S_{3K} to encrypt the second and third messages.

Appendix G. Meta-Review

The following meta-review was prepared by the program committee for the 2027 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

G.1. Summary

This paper analyses Matter's PASE, CASE, and CASE Resumption protocols through specification analysis, implementation auditing, symbolic verification, and attack demonstrations. It identifies gaps between Matter's stated countermeasures and the protection provided under specific compromise and commissioning scenarios.

G.2. Scientific Contributions

- Identifies an Impactful Vulnerability.
- Provides a Valuable Step Forward in an Established Field.
- Creates a New Tool to Enable Future Science.

G.3. Reasons for Acceptance

1) **Identifies an Impactful Vulnerability.** The paper shows that the setup passcode can be recovered from w0 alone, challenging Matter's T102 countermeasure under weaker compromise assumptions than it considers. The implementation audit also identifies failures to enforce security requirements, with responsible disclosure leading to patches.
2) **Provides a Valuable Step Forward in an Established Field.** Combining specification analysis, code auditing, and symbolic verification provides a systematic account of where Matter's security guarantees depend on deployment choices and implementation behaviour. The study connects concrete findings to Matter's own threat model and countermeasures.
3) **Creates a New Tool to Enable Future Science.** The reusable ProVerif models provide a foundation for further analysis of Matter's key-establishment protocols. They support examination of authentication and secrecy properties under explicit compromise assumptions, including the difference between full CASE and CASE Resumption.

G.4. Noteworthy Concerns

1) **Scope of the attacks.** Practical impact depends on prior compromise or passcode knowledge, an available commissioning window, and the supported recommissioning method. CitM additionally requires acceptance of failed device attestation and has not been evaluated on commercial devices.
2) **Evaluation of the OPRF mitigation.** The proposed OPRF-based extension remains a direction for future work. Its security, deployment feasibility, and computational and storage costs on constrained devices have not been evaluated.