What a Mesh: Unified Formal Security Analysis of WPA3 SAE Wireless Authentication

Author One
Affiliation
Email

Author Two
Affiliation
Email

Author Three
Affiliation
Email

Author Four
Affiliation
Email

***Abstract*—The latest WiFi security standard, IEEE 802.11, includes a secure authentication protocol called WPA3-SAE. The protocol is specified at two separate but linked levels: a traditional cryptographic description of the communication logic between network devices, and a stateful finite automata description that realises the former in a single device. Current formal verification efforts focus mainly on communication logic. We present detailed formal models of the protocol at both levels, provide precise specifications of its security properties, analyse machine-checked proofs in ProVerif and ASMETA, and show how they link. Particularly novel is the combination of the above two models, which enabled us to address several issues in the current IEEE 802.11 specification more thoroughly than would have been possible otherwise, leading to several revisions.**

1. Introduction

For years security experts, cryptographers, and security engineers alike have struggled through the process of interpreting large protocol specifications, with the intent of formalizing, analysing or implementing them, even when they present ambiguity. Research efforts from the formal methods community aimed at automating security analysis are part of a vast body of literature focused on transforming (often manual and error-prone) specifications into formal models that can be analysed for security properties [1], [2], [3], [4], [5], [6], [7], [8].

Since its introduction in 2018, WPA3 has aimed to enhance the security standards of WiFi networks established by WPA2. Nonetheless, recent works demonstrate how WPA3 remains susceptible to known vulnerabilities [9] that have been shown for some implementations. This work aims to complement previous works by presenting a formal verification of WPA3's security architecture, providing an evaluation of its actual security capabilities.

Traditionally, protocol verification has focused primarily on message exchanges, which, while effective at establishing verification methodologies [10], [11], [12], often neglects vulnerabilities that may arise from the behaviour of individual devices [6]. Such vulnerabilities are critical, as they may cause severe breaches when the protocol is implemented in real-world environments.

We focus on WPA3-SAE (Simultaneous Authentication of Equals), which is a security protocol specified in the latest IEEE 801.11 standard [13]. Its specification is divided into two separate but linked sections that reason at two different levels. The first section describes a traditional cryptographic description of the communication logic between two parties, which we call *Communication level*. This is opposed to a second section that describes how to realise the protocol in the first section as a stateful finite automaton in a single device, that we call *Device level*.

Our research introduces a dual-level verification approach that employs traditional Dolev-Yao symbolic analysis at the Communication level combined with a transition system analysis at the Device level. The two specifications are largely self-contained (even when linked), so we modelled them independently first, to discover that they were misaligned, leading to potential vulnerabilities in different yet compliant implementations of either of the two. More interesting in a secondary phase we combined the two models, feeding one with the other to improve the accuracy of our formalisms. This allowed us to propose several patches to the standard[1], most of which have been accepted and will appear in the next revision of the IEEE 802.11. Our results indicate that each model captures unique aspects of the protocol's vulnerabilities, with their integration providing new insights that significantly improve our understanding and patching of the standard.

Some of our findings can be linked to previous attacks on WPA3, confirming their effectiveness in identifying and describing new vulnerabilities. For example, the memory exhaustion attacks described by Vanhoef and Ronen in [9], share the same nature of deadlock situations that we have

1. Hyperlinks and references to documents are omitted here to keep anonymity, but a redacted version of them is provided in our artifact: https://doi.org/10.5281/zenodo.14168708. Alongside all of the models.

noticed at the Device level. These findings contribute to our development of more secure wireless protocols.

We summarize our contributions as follows.

1) We propose a novel, integrated approach to protocol verification that addresses both communication and device-level security;
   - as part of this process, we added equations to our models that capture the modular division operations involving the group exponentiation (Section 5.1.1), that are general and reusable in other contexts where such operations are used.
2) We apply this methodology to WPA3-SAE, uncovering over 20 new security issues (which have now been amended in the standard).
3) Finally, we provide validated solutions to these issues, improving the protocol's resilience against a variety of attack vectors.

The rest of the paper is structured as follows: Section 2 summarises work in this area and related literature. In Section 3, we introduce the Simultaneous Exchange of Equals (SAE). We then explain our methodology in Section 4 and provide details of our analysis of the WPA3-SAE formal models in Section 5, followed by our results in Section 6. We finally discuss our findings in Section 7 in relation to real-world WPA3 implementations and draw our conclusions in Section 8.

2. Related Works

The unification of multiple tools for the verification of protocols has come under substantial attention recently [14], [15], this is not altogether new as it was one of the core principles of AVISPA [11] a couple of decades ago, whose intermediate language is based on the AnB logic, and could not be used to capture enough security aspects of complex and modern protocols. Later on, the earliest work discussing unification for modern verifiers was in 2019, where authors combined the verification of a protocol translating from an intermediate language to both Tamarin and ProVerif [16]. However, such seminal work was lacking soundness of translation and the semantics of its language, to the best of our knowledge, was never completely formalized or studied; thus, it remains difficult to trust in the domain of formal verification of security protocols.

Successively, an independent work extended the security guarantees typically associated with protocol designs to their actual implementations [15]. This is achieved by instrumenting common cryptographic libraries and network interfaces with a runtime monitor. By focusing on runtime verification, the paper addresses the dynamic aspects of protocol security, ensuring that the protocols behave as expected not just in theory but also in real-world operational environments.

The most recent work advancing on the unification of protocol verification techniques presents a methodology for the modular verification of protocol implementations [14]. Its methodology leverages verification logics and tools, supporting a wide range of implementations and programming languages. The effectiveness of this approach is demonstrated through the verification of memory safety and security of various protocol implementations.

Probably the closest work to ours can be found in an early verification of TLS1.2 [17], where authors identified that the state machine is an important part of the implementation, and performed some verification on it. The verification process includes type-checking the state machine of the TLS protocol. Type checking is a method of verifying that the program adheres to the specified types, which in this context, relate to the correct sequence of operations and data handling in the protocol. This ensures that the state transitions in the TLS protocol adhere to the defined security specifications. Although this is undoubtedly a first step, authors have never linked the state machine to the upper layers of the protocol, as we do in our approach, and no further verification beyond type checking is performed. We argue these are not distinct steps, and indeed the verification of the state machine properties can lead to better analysis of the network protocol and vice versa.

We note that formal analysis is not the only way to analyse a protocol, and indeed, authors have already run security evaluations of similar protocols [9], [18], i.e., the DragonFly protocol, which is the predecessor of the SAE key exchange. Their work found several vulnerabilities, with a focus on side channel attacks [9] and communication layer attacks [18], in several real-world devices that implemented the key exchange. Their work contributed to a more robust successor that could introduce numerous changes - including security patches - that distinguish it significantly from its predecessor. Such influence can be read from the IEEE 802.11 standard, which was adopted in 2016 and expanded further in the latest 2020 review. Our work is complementary to theirs, as we focus less on implementations; indeed, we formalize the standard for symbolic analysis, identify (some of) the points of vulnerability and ambiguity that contributed to these issues appearing in implementations, and find new sources of potential vulnerabilities.

3. WPA3-SAE

WPA3 security is ensured using the Simultaneous Exchange of Equals (SAE) protocol, which is introduced to replace the Pre Shared Key Exchange (PSK) used in WPA2, which was known to be vulnerable to attacks [18]. SAE provides authentication at the Link layer referring to the TCP/IP stack. It was originally introduced in 2016 as part of IEEE 802.11 - IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems [19], with claims to resolve the previous vulnerabilities affecting PSK and WPA2. We focus our security analysis on the latest version protocol, following the specifications from IEEE 802.11w [13] when conducting our assessment.

3.1. The SAE Key Exchange

The SAE key exchange is a two-party protocol that takes two rounds of messages. Each round is symmetric so that we do not have a notion of an Initiator and Responder or of a Supplicant and Authenticator. Each side may initiate the protocol simultaneously such that each side views itself as the *initiator* for a particular run of the protocol. This design is necessary to address the unique nature of mesh basic service set [13]. Several variants of SAE are specified; nonetheless, we only focus on the variant of SAE adopted in the WPA3 protocol, especially the variant that operates in finite field cryptography[2].

The SAE protocol operates in $\mathbb{G}$, a common and public subgroup of $\mathbb{Z}_p^\star$ of multiplicative order q, where q is a Sophie Germain prime, i.e., $p = 2q + 1$, where the discrete logarithm problem is assumed to be hard. The protocol also uses H representing a hash-based message authentication code (HMAC). Two remote parties, Alice and Bob, share a common secret password from which they apply a transformation to calculate a corresponding *password element*, PE. Using PE, a secret element can be derived as described by the Key Derivation Function KDF, defined in the standard [13]. The (last) confirmation phase also defines a confirmation function CN, which is also defined in the standard. Both KDF and CN are calls to the hash function H. The SAE protocol runs in two rounds: the *commit exchange* and the *confirmation exchange*, as illustrated in Fig. 1 for the peer A, which communicates with a symmetric peer B.

3.2. The SAE Message Handling

Message handling in WPA3 is described in §12.4.8 of the standard, in terms of a state machine, see Figure 2, as interactions between three entities: 1) Station Management Entity (SME), 2) Parent Process (PP), and 3) Protocol Instances (PI).

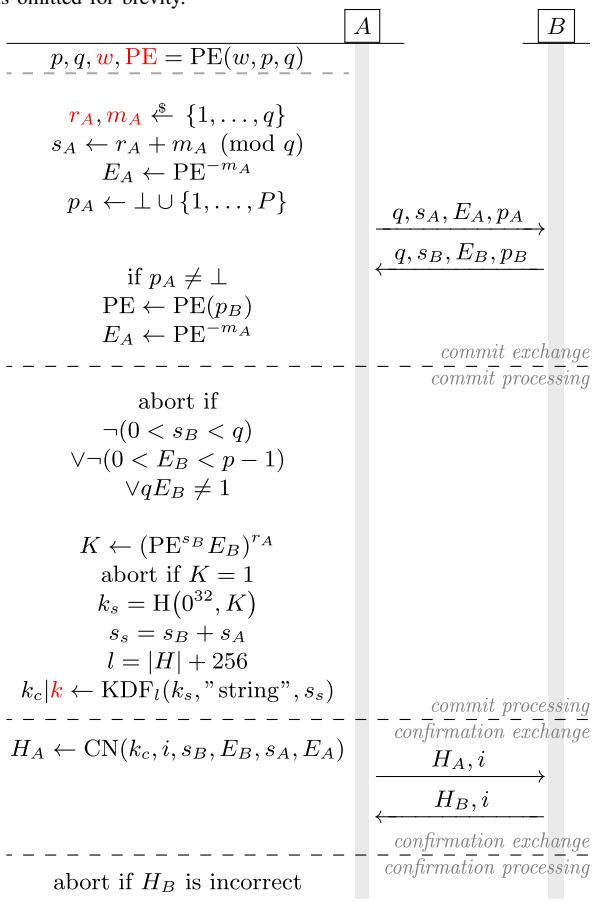
SME. This entity is a component responsible for managing various aspects of a wireless station or device. It is primarily concerned with managing the physical and medium access control (MAC) layers of the 802.11 protocol. SME provides an interface for higher-layer protocols to interact with the lower layers and handles tasks related to the wireless station's configuration, operation, and maintenance. However, this entity is not strictly described in the standard, which states:

> Some functions of the SME are specified in this standard.
>
> §6.1, pg. 314

Moreover, the description is fragmented and given incrementally in different parts of the documentation. This easily causes misinterpretation and misunderstanding.

2. Our models abstract the mathematical operations in the finite field, so that are analogous to the Elliptic Curve specifications.

Figure 1. SAE protocol in WPA3 [13] at Communication level. A and B share the secret password w and computed PE in private; $\mathrm{PE} \in \mathbb{G}$ and $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^\star$ of order q; i is a counter. B is symmetric to A and thus omitted for brevity.



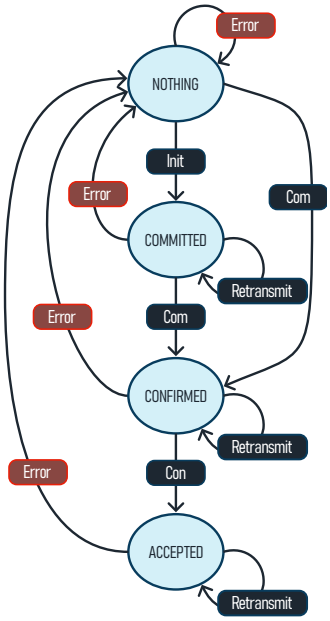
PP. The parent process is in charge of managing the database of the protocol instances (PIs). It performs a number of tasks, such as allocating and deallocating instances, and keeping track of their respective states. Additionally, it is responsible for routing incoming messages from the environment to the correct PI (allocating a new instance when needed). To accomplish these operations, it sends a suitable event to drive the state change of the PI. It also keeps the database updated based on the events it receives from both the SME and the PIs.

PIs. The processes that ultimately realise the protocol enact the behaviour specified as a state machine. State changes are triggered by messages and events received from the PP, but transitions fire upon the analysis of message content. Transitions execution is accompanied by actions that the PI executes before entering a state. Such actions consist of messages for the peer or generation of *error* or *completion* output events to be sent to the PP for subsequent deallocation of the instance.

Fig. 2 depicts a simplified version of a PI's state machine consisting of four states and having transitions labelled with the relevant events sent by the PP and able to trigger a

state transition (note that we omit information regarding the received messages that might prevent transition firing, and actions performed by the PI when a transition fires).

Figure 2. State machine describing the SAE protocol in WPA3 [13] at the Device level.



A PI is in the state Nothing initially, as a new instance, and finally, as a terminal state before being deallocated in case of error. Depending on their allocation, new PI immediately transitions out of the Nothing state to either Committed or Confirmed. Protocol instances that transition into the Nothing state shall immediately and irretrievably be deleted[3]. A PI enters state Committed when it receives the event Init from the PP and has sent an SAE *commit* message to the peer. In this state, the PI waits for the Com event from the PP and moves to state Confirmed having sent an SAE *confirm* when it receives the correct SAE *commit* message from the peer. The PI remains in this state if a Retransmission[4] event happens. When an unmanageable error occurs, a Del event is sent to the PP, and the PI moves to state Nothing to be deallocated. In the state Confirmed, the PI operates as in state Committed in case of events Del and Retransmission, and moves to state Accepted on the event Con from the PP and by receiving a peer's correct *commit* message. In the (final) Accepted state, events Retransmission and Del, if happen, are dealt with as in the previous states; otherwise, the PI has successfully concluded the process.

4. Methodology

We here detail the main contributions of our work, which are threefold, each with distinct approaches: 1) Unified Methodology, 2) Security Analysis of WPA3-SAE, and 3) Validated Solutions.

4.1. Unified Methodology

The unified methodology, at the Communication and Device level that we propose, concretely involves: a **dual-level modelling**, where we specify models that separately reflect the specification at the Communication and Device levels; a **level-specific analysis**, which exploits different formalisms and verification paradigms at the two levels; and a **unified analysis**, where we feed one model with the results of the other, to reach a single coherent description of the protocol, which is at the basis of our suggested verified patches. The formalism at the Communication level uses a well-established modelling technique, whose analysis is mechanized and runs on a state-of-the-art formal verification tool, ProVerif. Whilst the formalism at the Device level is based on the stateful analysis of state machines, whose verification is mechanized and runs on a state-of-the-art formal framework, ASMETA. This approach allows for a much deeper study of a protocol's security, allowing for greater insights into how well the protocol functions (thus allowing for better implementations). We leverage the strengths of each level of specification to cover a comprehensive range of aspects and verification goals contextual to such levels.

At the Communication level, we use the π-calculus as specification formalism, Horn clauses resolution as a verification mechanism, and ProVerif [20] as a tool. At this level, we analyse properties common to other key exchange verification processes. The properties we verified in the π-calculus (presented formally in later sections) are:

- **Correctness** - Verifies that after finalising the exchange, both parties share the same key.
- **Authentication** - Checks that the entities involved in the communication are who they claim to be, ensuring that messages are exchanged between legitimate and verified parties.
- **Key secrecy** (SK) - Ensures that the shared session key remains confidential and is only known to the participating parties.
- **Password element secrecy** (SPE) - Guarantees that individual elements derived from the password during the exchange are not exposed, even if other parts of the system are compromised.
- **Perfect forward secrecy** (PFS) - Assures that session keys cannot be compromised even if the long-term private keys are exposed in the future.

At the Device level, we use state-based transition systems as our specification formalism (specifically, the Abstract State Machines [21], [22]), which are efficiently supported through the tool-set ASMETA [23]. ASMETA has been used for swift model editing (with AsmetaL notation), validation (with AsmetaV [24] and its notation Avalla), and verification using temporal logic properties utilizing model checking (with AsmetaSMV, which maps AsmetaL models to the model checker NuSMV [25]). Our analysis focuses

3. This relevant information is reported only in section 12.4.8.2.2 of [13]

4. This event groups several wrong (but manageable) error cases that can happen, e.g., wrong order of the received messages, wrong received group, etc.

on the compartmentalised analysis of individual agents, i.e., PIs, PP, SME, each of which has its own Abstract State Machine (ASM) that communicates with the others via events.

At this level, we concentrate on the operational correctness of message handling, synchronization mechanisms, queue implementations, and state reachability within the protocol. Since security and authentication properties have been handled by verification at the Communication level, where the messages' content is thoroughly checked, during our analysis of the agents' machines, we focused on the following verifying properties, listed along with their corresponding CTL formulas:

- **Safety** - bad configurations do not happen; in CTL, $AG(\neg\phi)$, which means that the condition ϕ must never be violated globally.
- **Reachability** - desired configurations are reachable; in CTL, $EF(\phi)$, which states that there exists a future state in which the property ϕ is true.
- **Deadlock-free** - it is always possible to exit from given configurations; in CTL, $AG(EX(\phi))$, indicating that in every state, there is an immediate successor state that satisfies the property ϕ.

Further CTL formulas have been stated to evaluate corner cases, such as *edge safety* property checking for the possibility of agents' failure to receive events caused by a lack of RAM or caused by simultaneous reception of events from multiple agents. One additional step at the device level that provides formal guarantees of the model's compliance with the specifications is to create scenarios using the AsmetaV tool. This approach ensures that the state machine remains consistent, enabling the identification of specification errors without relying on formal properties. As a result, this method shortens the verification time and enhances confidence in the model's completeness and correctness. This advantage is significant and, if possible, not easily achievable with languages that typically model the Communication level [5].

4.2. Dual-Level Security Analysis of WPA3-SAE

Following the model formalism, verification and analysis were conducted using ProVerif as the verifier of the models in π-calculus and ASMETA as the validator and model-checker of the stateful representations.

Diverging from the usual independent status quo, these results were then collated and unified in a refinement stage. Whilst we may wish to treat individual layers of the communication stack as separate, they are inevitably interlinked, and as such, it is sometimes the case that when a vulnerability is found at a specific layer, it might be that it is incorrectly handled at a different layer (for better or for worse), leading to false attacks. In theory, this is not always a problem, indeed finding errors in protocol specifications is a worthwhile exercise even in practice, even in those cases when this does not lead to an attack that can be verified from a compliant implementation [6]; however, for a real-world implementation check, having this extra knowledge is a crucial advantage.

4.3. Validated Solutions

The final stage was to validate the results as well as patch the attacks. This effort was threefold: 1) formally verify that our proposed solutions fixed the vulnerabilities by running the tool on updated models; 2) coordinate with IEEE 802.11 Working Group - those in charge of the changes to the specification - to update the standard to reflect the new verified patches; and 3) check the largest open-source implementation of WPA3 (`hostapd`), to compare with the interpretation from the point of view of developers, and see if and how they handled the issues we have found.

Concerning the second stage, we were added as designated experts to the IEEE802.11p Working Group. After explaining our points in several meetings and attending panel meetings, almost all our corrections were accepted and introduced into the standard (19/25 accepted) [7]. We would like to note, that although six proposed changes were not accepted, they were acknowledged as distinct issues, but deemed too unlikely to warrant a re-specification. This is another difference between formal analysis and the real world, where in the formal domain any potential threat is an error, in the real world, we found, based on these discussions, that sometimes performance and usability are valued above a potential, unlikely case of threat.

5. Formal Models of WPA3-SAE

As per phase one of our methodology, we first modelled the Communication and Device levels of the protocol.

5.1. Symbolic Models at the Communication Level

As can be easily seen, the π-calculus code inside the boxes in Fig. 3 is the part modelling the protocol scheme depicted in Fig. 1.

We highlight the symmetric nature of the protocol by allowing both both processes to write their first message to the channel before reading from it: in other words, the non-deterministic symbolic execution will explore both cases when either peer will read from the channel first.

The π-calculus code outside of the box serves to model two aspects: first, to implement the usage of the same password element PE; and second, to model the usage of the key k after the key exchange for secrecy properties. In detail, the first line looks for the password element PE in

5. Indeed, completely valid interpretations of the specification have resulted in wrong verification results in previous work [26], due to lack of conformity checking

6. A potential downside of this however is that these lack of relationship between the error and the fix may lead to the fix being then deleted in later versions - as happened with WPA2 when moving to WPA3.

7. REDACTED FOR ANONYMITY, we provide the anonymised working group minutes in our artefact: https://doi.org/10.5281/zenodo.14168708.

Figure 3. The vanilla description of the SAE protocol.

```
1  P_L ← get t_p(=L, =R, =pw, PE) in
2        νr_L.νm_L.
3        let s_L = r_L + m_L in
4        let E_L = PE^{-m_L} in
5        let c_L = (s_L, E_L) in
6        out(c, c_L); in(c, c_R);
7        let s_R, E_R = c_R in ;
8        let K = (PE^{s_R} E_R)^{r_L} in
9        let k_s = H(0^32, K) in
10       let s_s = s_L + s_R in
11       let k_c = kcf(k_s, "SAE", s_s) in
12       let k = pmk(k_s, "SAE", s_s) in
13       νi_L.  // send-confirm
14       let H_L = CN(k_c, i, c_L, c_R) in
15       out(c, H_L, i_L); in(c, H_R, i_R);
16       out(c, enc(k, m));
```

the table t that has been computed by the participant, either Leftmost L or Rightmost R, and intended to be used with their peer. If t contains a suitable PE to communicate to the other party, the participant continues; otherwise, it aborts. It is worth noticing that, however, this would not stop an active attacker from their attempt to tamper with messages or craft new ones. Finally, the last line is not part of the scheme and has been artificially added to verify the secrecy of the shared key k through the privacy of the message m. We model the counter `send-confirm` as the nonce i to simplify the verification efforts. This assumes that the counter is unpredictable; though, we observe that it is sent in clear along with H_L (or H_R) and thus it is known to the adversary.

The two parties. The SAE protocol illustrated in Fig. 3 is constituted by two participants. We call one the Leftmost, L, and the other the Rightmost, R, for the convenience of naming in our model; however, we stress that given its symmetric structure, the protocol can be initiated by any device. All models of the protocol are variation of the same vanilla specification, where we abstract out the pairwise master key as pmk and the EAPOL-Key confirmation key kcf.

The pre-shared password. A table t_p of passwords is filled with all password elements PE that would be calculated by the participants before engaging the protocol, i.e., $\mathrm{PE} \in \mathbb{G}$ is the secret group generator for L and R implemented with finite field cryptography.

From the point of view of the symbolic protocol design, picking up from a pool of passwords and then computing the password element (or generator) is the same as having directly shared the secret password element.

The models in Fig. 3 are slightly different from the specification. In particular, the first two lines and the last line of both processes are unexpected and are syntactic sugar to better capture the original protocol in our formal definition.

The protocol is supposed to run asynchronously, i.e., the message order is irrelevant, and the responder may actually behave as the initiator. However, this is only true for the first phase, since in the second phase the parties need to know who sends the double and who sends the single hash of the key. This behaviour is formalised in the key-exchange phase by letting both parties output before reading, but this cannot be done in the key confirmation phase. In addition, our model does not check if the incoming message is in the range $\{2, \ldots, p-2\}$. Due to the limits of the tool, one cannot write the group theory in ProVerif to model such a check. All the variants of the SAE protocol we have implemented are based on varying the model in Fig. 3.

The main process. Symbolic analysis generally allows for the security analysis of parallel execution of protocols. To this end in π-calculus, one can implement the main process, P, as the parallel execution of two processes: the Leftmost process P_L and the Rightmost process P_R, whose details model the protocol behaviour for each party. Due to its symmetric nature, any peer can initiate the protocol, so the implementation could just be a single self-composed process that will be used by both parties. Unfortunately, such a naive, direct model of the SAE protocol would easily produce *false* attacks where an initiator would speak to itself. To avoid this unwanted behaviour, a solution is to explicitly support the session between the two parties. However, the session, denoted as sID, is not private information and is known to the attacker. To model that, we simply push it to the insecure channel c, i.e., $\mathsf{out}(c, sID)$. Before calling the processes P_L and P_R, we establish which party runs P_L and which runs P_R. Doing this, we capture the ability of honest parties to engage with the protocol with either algorithm, whose structure is anyway mirrored. This choice requires us to make sure that the two parties would not engage in the protocol if they do not share the password element. For this reason, we also added an environment process P_P, which is in charge of inserting shared password elements into a table that will be accessed by P_L and P_R but cannot be accessed by the attacker.

Several security properties are based on writing custom events into the execution trace. In order to record such events, we have a special process P_A, which records some variables in the meanwhile the protocol is executed, at the convenience of the security properties that we want to analyse, i.e., correctness. In practice, the process P_A would collect information from additional tables filled with terms whose content is ultimately put together recording a single event of *agreement* of terms generated by different entities. For those security properties which do not require tables to record events, the process P_A will be simply empty, i.e., $\mathbf{0}$.

Finally, the main process P that the tool checks has the following structure:

$$\begin{aligned} P_{SAE} \leftarrow\ & \nu sID.\ \mathsf{out}(c, sID); \\ & (P_L(L, R, sID) \mid P_R(L, R, sID)) \\ P \leftarrow\ & P_P \mid P_{SAE} \mid P_A, \end{aligned}$$

that informally reads as the composition of the setup of passwords, the protocol, and the final agreement of terms.

Modelling attacks and security properties. Some attacks or security properties are based on the inspection of the traces of execution of the protocol. Each trace t in the space of all infinite traces T is a potentially infinite sequence of generic elements. A trace is a list that reflects a possible expansion of all combinations of the concurrent executions of a process (a protocol)[8]. A trace is implicitly temporally ordered, in the sense that $i < j$ implies that $t[i]$ (the i-th element in the trace t) has been recorded before $t[j]$ during the expansion. Attacks and security properties are both statements that relate to the traces of execution. While to demonstrate an attack, an example is sufficient to show, conversely, a security property must be a statement (a theorem) over those traces that capture desirable behaviour of the protocol with global validity.

For some security properties, artificial events has to be injected in the definition of processes. An event e can be recorded as an element in a trace t. We denote $e \in t$ to say that an event appears in the trace t. For example, if we inject one event, e_{BEGIN}, that records the beginning of the protocol, and another event that records its end, e_{END}, then we can capture authentication by requiring that the presence of e_{END} in *all* traces t implies the presence of e_{BEGIN} in t before e_{END}:

$$\forall t \in T.\ t[j] = e_{\text{END}} \Rightarrow \exists i < j.\ t[i] = e_{\text{BEGIN}}.$$

We simplify this notation by removing indices, we can write equivalently $\forall t \in T.\ e_{\text{END}} \in t \Rightarrow e_{\text{BEGIN}} \in t \wedge e_{\text{BEGIN}} < e_{\text{END}}$. Precise relations that capture security properties, and that follow from the above basis, will be provided in Section 6 as they are required.

5.1.1. Handling Unsupported Mathematical Operations. The reader who is more acquainted with the π-calculus must have already noticed that the mathematical theory behind the operation $K = (\mathrm{PE}^{s_R} E_R)^{r_L}$ in Figure 3 is not supported by ProVerif. ProVerif only supports the exponential operation, whose commutativity can be defined as described in its manual when reasoning about the Diffie-Hellman Key Exchange protocol. Conversely, multiplications (and additions) can only be partly defined. They cannot be fully supported for the simple reason that commutativity and associativity would induce the reasoning core to infinite expansions, thus to non-termination. If the protocol allows it, the designer can model only part of them: we model commutativity but do not need associativity. This choice is not unusual; for the sake of another example, we reference the π-calculus models of recent models of TLS [27].

An additional complication occurred in the process of defining the division as the inverse operation to the multiplication: modelling the division is *required*, otherwise the K cannot be reconstructed by both peers and *correctness* and *secrecy* cannot be reasoned about. To overcome the above-known limitation, we modelled the division modulo p, denoted as $\div_p$, as an *extended destructor*. Such destructors are defined by subexpressions; then, "each subexpression [...] is rewritten by trying the rewrite rules of [the destructor] in the order given [...], and applying the first applicable rewrite rule." [12].

In detail, we first tried to implement $\div_p$ as a function and an equation; however, ProVerif could not handle its equations, as the tool could not prove their convergence (thus termination). Our next step was then to axiomatically add their convergence; alas, ProVerif does not allow adding such axiomatic behaviour (and skip termination check) if more than one equation uses the same function; in our case, the *blocking* function is the standard and (the only) documented equation to be used to model group exponentiation from the ProVerif manual.

Finally, were successful with the following *extended destructor*:

$$\begin{aligned} \div_p \equiv\ & \big[\forall a \in \mathbb{G}, x, y \in \mathbb{N}.\ a^{x+y} \div_p a^y = a^x, \\ & \forall a, b \in \mathbb{G}.\ a \div_p b^{-1} = ab, \\ & \forall a, b \in \mathbb{G}.\ ab \div_p b = a\big]. \end{aligned}$$

Clearly, those rewriting rules are far from being able to define the division comprehensively but are reducing from the left to the right (to reach termination), and enough to reconstruct the final key. They are even slightly more expressive than what is strictly required to reconstruct the key; this is useful to enrich the capability of the adversary to enjoy (marginally) stronger guarantees.

5.2. Symbolic Models at the Device Level

Abstract State Machines (ASMs) are a state-based formal method that enhances Finite State Machines (FSMs) by substituting unstructured control states with algebraic structures. These structures are defined as domains of objects that have functions assigned to them. State transitions occur through the execution of transition rules.

During each computation step, all transition rules are executed in parallel, resulting in simultaneous and consistent updates to several memory locations. These locations are defined as pairs of (function-name, list-of-parameter-values), which means the interpretation of functions changes from one state to the next.

Updates to locations are expressed as assignments in the form of loc $:= v$, where loc represents a location and v signifies its new value. Among various rule constructors utilized, some relevant for our purposes include constructors for guarded updates (such as `if-then` and `switch-case`), parallel updates (`par`), sequential actions (`seq`), nondeterministic updates (`choose`), and unrestricted synchronous parallelism (universal quantification `forall`).

Functions that are not updated by rule transitions are termed *static*, whereas those that are updated are called *dynamic*. Dynamic functions can be further classified as

8. Some tools can reason about an unbounded number of concurrent executions, others are capped.

monitored (read by the machine and modified by the environment) or *controlled* (read and written by the machine). The ASMETA framework allowed us to model the state machines of the PP, the PIs and the SME. As an example, in Fig. 4, we show an excerpt of the ASM rule of the PP program called `r_SME_Signal`[9]. The rule is part of the model used for validation; more specifically, it describes the handling of events produced by the SME agent and consumed by the PP agent. The INITIATE and KILL events affect the allocation of new instances of the PI agent, respectively creating new ones and forcing their removal from the database.

Figure 4. Allocation of a new instance in the state Nothing after receiving the event INITIATE.

```
rule r_SME_Signal=
  forall $mac in Mac_List with signal_to_Parent_Process($mac)
          !=NOSIGNAL do
   if(database_instance_counter($mac)=0 and
          signal_to_Parent_Process($mac)=INITIATE )then
    extend ProtocolInstance with $pi_first do
      par
          state_MAC($pi_first ):=NOTHING
          protocol_Instance_Event($pi_first):=INIT
          database_instance_counter($mac):=
              database_instance_counter($mac)+1
          database_instance($pi_first):=true
          parent_Process_Event($pi_first):=NOEVENT
          protocol_Instance_Association($pi_first):=$mac
          open:=open+1
      endpar
      ...
   endif
```

The producer-consumer logic outlined in the rule operates as follows: The Process Publisher (PP) first checks, using the `forall` primitive, for any signals sent by the Service Management Entity (SME) indicating an event related to the selected MAC address from the `Mac_List` domain, which includes all MAC addresses under analysis.

Next, the protocol requires checking the cardinality of MAC instances present in the database. Specifically, it verifies whether the count of instances for the given MAC address is zero (`database_instance_counter($mac)=0`). This verification occurs when a request to allocate a new instance is initiated via the INITIATE event (`signal_to_Parent_Process($mac)=INITIATE`).

When this condition is satisfied, the PP employs the `extend` primitive to add a new Protocol Instance (PI) to the `ProtocolInstance` domain. The newly created PI is initialized to the NOTHING state(`state_MAC($pi first ):=NOTHING`). An INIT event is then sent to this PI(`protocol_Instance_Event($pi first):=INIT`), and its corresponding database information—such as the number of occurrences and the binding between the allocated PI and the MAC address(`protocol_Instance_Association($pi first):=$mac`)—is updated accordingly. Finally, the PP is set to listen for new events by setting to `NOEVENT` the controlled function `parent_Process_Event($pi first))` and increments the total number of open connections in the `open` variable. Associated rules follow for all the other transitions from Figure 2.

The model created by combining the programs of all agents in the protocol was crucial for validating compliance with the requirements outlined in the standard specification. This model allowed for a quicker and more efficient refinement process. It soon became apparent that this approach is versatile and effective in identifying inconsistencies or gaps in the specification. For example, it revealed issues such as the use of incorrect or previously unmentioned names, like in the thirteenth errata, where the variable `SYNC` was used instead of the `big(sync)` event, which describes the maximum number of allowed synchronizations. This approach also relieves the verification part of properties that can be disproved by deploying a scenario, allowing the user to concentrate on more complex properties.

Unlike the communication part of the specification, many inconsistencies and incompleteness were found in the official documentation for the Device level process. The complete final model has been obtained by a model refinement process using using Avalla [24]. An example scenario using AVALLA may be seen in Figure 5.

Figure 5. Scenario based testing of ending of COM event when the PI is in ACCEPTED, simplified first step out of three.

```
scenario PI_Accepted_SYNC
load SAE_20_scenari.asm
...

set select_mac_sme_signal :=MAC1;
set signal :=NOSIGNAL;
set select_mac :=MAC2;
set message_from_peer(MAC2) :=EMPTY;

exec extend ProtocolInstance with $pi do
  par
    protocol_Instance_Association($pi):=MAC1
    database_instance($pi):=true
    state_MAC($pi):=ACCEPTED
    database_instance_counter(MAC1):=1
    save_PI:=asSequence(ProtocolInstance)
    start_Timer($pi,TIMER0):=false
    sync($pi):=6
    sc($pi):=0
  endpar
;

step
...

set select_mac_sme_signal :=MAC4;
set signal :=NOSIGNAL;
set select_mac :=MAC1;
set message_from_peer(MAC1) :=CONFIRM;
...
```

Building scenarios to validate the requirements of the

9. All the PIs execute the same machine and each PI identifies *self* as itself.

standard was useful in finding inconsistencies and errors in our model (and the standards specification). Our first model incorporated all three agents (SME, PP and PI) and this model focused on understanding the agents' interaction and learning whether the rule sets defined for each agent meet the requirements. Upon the initial phase of model validation, using AVALLA, the model refinement step was performed using an iterative approach deriving insight from the finding of the result of in which we discovered some inconsistency in the specification. In Figure 5, we can see a simplified scenario as the protocol steps through the states and updates the state variables (according to the spec). This simulation approach allows for a quick test of coherence prior to running the more expensive verification steps. We verified our model by using AsmetaSMV, for verification purposes, the SME model was not considered.

6. Findings and Results

As introduced in Section 4, the WPA3-SAE protocol is specified in the traditional way of cryptographers, through explaining how messages are exchanged (Communication level), and the logic behind them that aim to validate security properties. In addition to this, a realization of such protocol is also described as a state machine, which is supposed to run in a single device (Device level). Such additional description focuses on state transitions and interface behaviour, prioritizing properties that relate to operational correctness within the device's environment.

We created models to capture the specification at two different levels - the Communication level, which is analysed through the lens of a Dolev-Yao attacker [28], and the Device level, which investigates the safety and completeness of the machine states of the agents involved in the protocol.

As expected, independent modelling efforts could capture different aspects of the WPA3-SAE protocol, detailed in Section 6.1, as well as some common aspects, detailed in Section 6.2. Interestingly, combining insights from both verification tasks allowed us to refine one or the other model to better understand the nature of the vulnerabilities and misalignments that we found. Consequently, findings in the verification in one level suggested ways to patch issues in the other level; for example, from the analysis of correctness at the Communication level, we could suggest how to patch the state machine modelling the Device level, something greatly useful for applying the most appropriate patch. We detail these *combined* findings in Section 6.3.

Table 1 reports the classes of checked properties and the verification results. In the following, only a relevant sample of these properties is presented.

TABLE 1. SUMMARY OF RESULTS ON FORMAL VERIFICATION OF SECURITY PROPERTIES IN PROVERIF (LEFT) AND ASMETA (RIGHT).

	Communication level					
	CO	**SA**	**WA**	**SK**	**SPE**	**PFS**
IEEE 802.11:2020	◒	○	○	●	●	●
Patched models	●	◓	◓	●	●	●
	Device level					
	SF	**DL**	**ES**	**RC**		
IEEE 802.11:2020	○	○	○	○		
Patched models	●	●	●	●		

Legend. Correctness (CO), Strong/Weak authentication (SA)/(WA), Key secrecy (SK), Password element secrecy (SPE), Perfect Forward Secrecy (PFS); Safety (SF), Deadlock (DL), Edge Safety (ES), Reachability (RC). **Outcomes**: (●) - verified, (◒) - some cases are verified, (○) - attacks found, (◓) - no attacks found, yet a final security proof cannot be automatically generated.

6.1. Independent findings: Replay Attack and Deadlock

6.1.1. Replay attack. The verification of the design of WPA3-SAE in the latest standard [13, §12.4] highlighted a vulnerability in the logical flow that led to breaking its authentication in our model.

On the basis of the vanilla code illustrated in Fig. 3, authentication of L in R (mirroring the process L) can be captured through the artificial injection of two events, e_i in P_L and e_f in P_R. Informally, e_i signifies the belief of L of having started an authentication process with R, and e_f the belief of R of having terminated an authentication process that must have been started by L, i.e., R believes that it is communicating to a genuine L. The standard way to capture authentication in the symbolic model is through verifying a correspondence of events in the traces of executions. In particular, if for all the traces t of the symbolic execution of P (defined in Section 5.1), the presence of the *final* event e_{F} in t is always after a single *initial* corresponding presence of e_{I} in the same trace, then we have verified authentication[10]. This is called a correspondence of events and can be formally described as

$$\forall t \in T.\ e_{\text{F}} \in t \Rightarrow\ !\exists e_{\text{I}} \in t \wedge e_{\text{I}} < e_{\text{F}}.$$

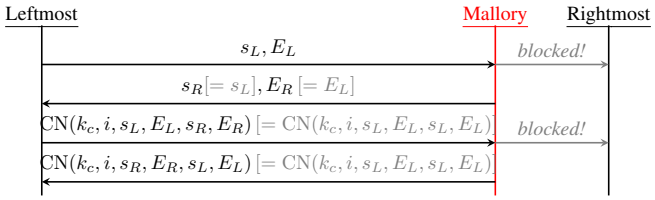
Strictly following the specification of the SAE protocol, the formal tool ProVerif [12] is able to reconstruct the flow of an attack as a counterexample of the property described above. By inspecting the reconstruction, we noticed that it is *simply* carried through blocking commit and confirm messages from L to R, when L initiates the protocol, and finally reflecting back to L its own messages. Fig. 6 shows the mathematical operations for which the replay of messages is (mathematically) acceptable.

The patch to this specific attack would be that of discarding messages with the same s_L, E_L or both. In our model, we can add the guard $c_L \neq c_R$ just after line 7 of the model of P_L described in Fig. 3 (and likewise for P_R which is omitted). That guard would allow the rest of the model to run and would stop in the case of a replay, i.e., $c_L = c_R$. With our patch, the same verification in ProVerif cannot find any attack on authentication any more.

An interesting note is that the realisation of the protocol as a single-device the state machine [13, §12.4.8.6.4] ignores

10. For mutual authentication, we need to inject other two analogous events inverting the processes.

Figure 6. Replay attack at the design level specification of the WPA3-SAE protocol in IEEE 802.11:2020, the variant that uses finite field cryptography.



replayed commit messages (allowing $s_L = s_R$ if $E_L \neq E_R$ and vice versa), de facto evidencing awareness about this attack. However, by deviating from the protocol specification, the state machine can compromise authentication, as the considerations at the Device level prioritize properties that relate to the operational correctness within the device's environment. It becomes evident that our comprehensive verification approach across multiple levels plays a crucial role in ensuring alignment with the intended protocol behaviour, thereby reducing the risk of insecure and buggy implementations.

6.1.2. Deadlock. Requirements in the WPA3 SAE standard include safety properties stated in natural language that can be translated into CTL (safety) properties and checked on the ASM model to ensure its robustness and correctness. E.g., the specification states that:

> For any given peer identity, there shall be only one protocol instance in the Committed or the Confirmed states.
>
> §12.4.8.6.1

This could be expressed by the following CTL formula stating that globally (AG) does not exist a state where two PIs (pi_1 and pi_2) have the same *mac* (i.e., the same identity) but incompatible *state* according to the specifications.

$$AG\Big(\neg\big(\mathrm{mac}(pi_1) = \mathrm{mac}(pi_2) \wedge \big((\mathrm{state}(pi_1) = \mathsf{Committed} \wedge \mathrm{state}(pi_2) = \mathsf{Confirmed}) \vee (\mathrm{state}(pi_1) = \mathsf{Confirmed} \wedge \mathrm{state}(pi_2) = \mathsf{Committed}) \vee (\mathrm{state}(pi_1) = \mathsf{Confirmed} \wedge \mathrm{state}(pi_2) = \mathsf{Confirmed}) \vee (\mathrm{state}(pi_1) = \mathsf{Committed} \wedge \mathrm{state}(pi_2) = \mathsf{Committed}) \big)\big)\Big)$$

Since this property requires checking the internal states of multiple PIs, it was verified against the PP model since PP has information on the current states of the PIs. The result of the property detects no attacks.

However, the standard documentation does not mention other relevant properties, such as the absence of deadlock, necessary to guarantee the completeness of error-handling cases. During the analysis of the absence of deadlock in the Committed state, we discovered, through the following failing property, which states that does not exist a configuration where the PI (pi) is in *state* Committed and an error occurs (*fail* holds),

$$AG(\neg EX(\mathrm{state}(pi) = \mathsf{Committed} \wedge fail))$$

that the standard does not handle the error in case of receiving a wrong commit message. This not-handled error causes the PI's state machine to deadlock, with no chance for the PI to be deallocated by the PP.

The standard specifies that a timer `T0` could have mitigated the deadlock by sending a Del event when it expired. However, the timer is deactivated when the PI is in the Committed state, and it checks the content of the Commit message, preventing the attack mitigation.

The deadlock is even more severe due to the safety property mentioned above because the attack can be easily executed by a malicious agent. Indeed, as a result of this deadlock, the PP is unable to create any new instances with the same MAC address as the one associated with the deadlocked PI. As a result, this may lead to a Denial of Service (DoS) attack on the affected peer, making it unable to connect to the network. The attacker could also exploit this vulnerability to cause a peer to run out of memory, as it is unable to deallocate all instances that ended up in the aforementioned error state. The last attack exhibits a similar pattern to the one described by [9], but it exploits a different vulnerability. The attack can be easily solved by adding, in the state machine, two transitions from the Committed state to the Nothing state. These transitions must involve sending a Del event to the PP to signal it to deallocate the PI.

6.2. Common findings: Correctness violation and Stall on bad password identifiers

Both the analyses of the two levels could model common issues with a novel feature that has been introduced in the revision 2020 of IEEE 802.11 [13]: the possibility of using multiple passwords. This change is meant to enhance security by resisting dictionary attacks, providing (selective) forward secrecy, allowing user flexibility, preventing credential sharing, facilitating secure connections for IoT devices, adapting to evolving threats, and improving overall usability. To enact this, an SAE entity can *require* its peer to use a specific password by sending a password identifier:

> If the peer's SAE Commit message contains a password identifier, the value of that identifier shall be used in the construction of the password element for this exchange.
>
> §12.4.5.4

And failure if no password maps from that password identifier. We could model the specification of correctness (from slightly different angles) in all the analysis levels, Communication and Device levels. In both we found that correctness was violated: on one hand, the formal verification at the protocol design level showed a case where peers would use a different password and thus, cannot complete the protocol; on the other hand, the formal verification of the safety property on the state machine showed that an unrecognised password identifier would lead to a stall where, again, the protocol could not complete.

6.2.1. Correctness violation. We implemented the exchange of password identifiers and their specified behaviour in ProVerif and found that this new feature breaks the security property of correctness in some unhandled cases. On the basis of the code illustrated in Fig. 3, we modified line 6 of P_L to $\mathsf{out}(c,(c_L,p_L));\mathsf{in}(c,(c_R,p_R))$, where p_L and p_R are the requested password identifiers. An analogous change is made for P_R. We have three cases: if $p_R = \perp$, a password identifier is not requested; if $p_R \neq \perp$ but p_R is not valid, then we abort; otherwise if p_R is a valid identifier, the password element PE' depending on p_R is used accordingly. This behaviour is captured by the following code:

$p_R \neq \perp$
  get $t_p(=L,=R,=p_R,\mathrm{PE}')$ in — fails if p_R not found
  let $E_L = \mathrm{PE}'^{-m_L}$ in — recalculate message
  $\mathsf{out}(c,(s_L,E_L))$; — re-commit

where a missing entry for an invalid p_R in t_p for L and R would automatically fail. Then, correctness is captured as a reachability property of an event e_{CORR} at the end of the protocol that includes the exchanged key k. The two processes P_L and P_R would write their entity names, the session and the exchanged key, each in their own table, t_L and t_R respectively, that cannot be read by the adversary but it is shared among processes. Then we instantiate the additional process P_A with an event that e_{CORR} that collects information from both tables. Formally, we first inject the table writing operation

$$\mathsf{insert}\ t_L(L,R,s,k) \qquad \text{and} \qquad \mathsf{insert}\ t_R(R,L,s,k)$$

just after line 7 of the model of P_L described in Fig. 3. Then, for all sessions s, pairwise master keys k, we require that the reasoning core is able to show a trace t where the event e_{CORR} is recorded and is such that two honest participants agree on their identities, the password, and the key.

$$\forall\, s,k.\ \exists\, t.\ e_{\text{CORR}}(A,B,s,k,A,B,s,k) \in t.$$

where A and B are (the only) honest parties, s is the (common) session ID, and k is the pairwise master key. If such an event is reached, i.e., e_{CORR} is found in t, then there exists a run of the protocol in which the two parties have authenticated each other and they have correctly exchanged the same session key. We firstly attempted to write this formalism in a single model. Unfortunately, ProVerif could not be conclusive (`could not be proven`). We managed to automate the process in ProVerif by hardcoding all different cases splitted into independent models. The only cases left out are those where the two peers do not share a requested password, as it was trivial in ProVerif to see that, as expected in such cases, the peers could not establish the communication. All cases are illustrated in Table 2. We remark that the semantics of the **set-set** case with different identifier was absent in the specification, and it has been agreed upon by the IEEE 802.11 Workgroup in light of our input.

In our analysis of correctness, if ProVerif is unable to reach e_{CORR}, then a key cannot be exchanged. In the cases where it should, it violates the security property of correctness. One such case is when both P_L and P_R require the usage of a specific password through a valid but different password identifier. In such a case, the password element PE will be different between the two peers, and they will not be able to verify the peer confirmation messages. The situation can be remarkably relevant in meshes, where the possibility of two peers initiating simultaneously can be frequent, and about which WPA3-SAE is particularly focussed:

TABLE 2. EXPECTED BEHAVIOR OF THE WPA3-SAE PROTOCOL WHEN PEERS EITHER OMIT OR SPECIFY (SET) A PASSWORD IDENTIFIER CORRESPONDING TO A PRE-SHARED PASSWORD.

	omit-omit	**set-omit**	**omit-set**	**set-set**
same identifier	●	●	●	●
different identifier	○	●	●	○

Legend. (●) - SAE should success, (○) - SAE should fail.

> SAE shall be implemented on all mesh STAs to facilitate and promote interoperability. §12.4.1

In practice, meshes treat all peers uniformly in the network, yet certain peers might possess multiple passwords tied to distinct profiles that they can seamlessly switch between. Again, we notice a deviation of the state machine to the protocol specification, as two peers starting the protocol with two different password identifiers would fail the authentication. Even if this *seems* just a patch in the wrong place, it is not and that is why: first of all it is coincidental, as there is no justification for such a deviation, but more importantly both peers would act legitimate and simply see the authentication failing, so they would just try again. In other words, there is no mechanism to prevent both peers from attempting the (failed) authentication repeatedly. Additionally, as the map of password identifiers is *not bijective*, they could have mapped to the same password with a different index for some reason, allowing the protocol to end correctly.

We can patch correctness requesting that, upon reception of the password identifier, a peer would *not* reply with a different one. In the case when both peers initiate and request for a different password identifier that is valid, then they will consider this a violation and fail. With this patch, the problematic case can be formally verified for correctness. Importantly, the aspects related to the semantics of multiple passwords go much beyond fixing correctness at the design level and requires further analysis that is not captured by our formalisation.

Finally, we stress that our model derives from an intuitive interpretation of missing specifications on how to handle password identifiers at the design level. In Section 6.3, we will examine how the outlined specifications, supported by formal verification, address the missing specifications.

6.2.2. Stall on bad password identifiers. In the documentation of the standard, the description of the multiple

branches of the state machine capturing the WPA3-SAE protocol behaviour is given in natural language and leaves out important details. This can result in ambiguity and interpretation, leading to serious vulnerabilities or attacks during implementation. For instance, when we modelled the PI state machine with ASMETA, we discovered an attack that was caused by unclear instructions on how the principal PI should handle an error message. The vulnerability lurks in the protocol requirement describing a case of failure:

> If so and there is no password associated with that identifier, *BadID* will be set and the protocol instance will construct and transmit an authentication frame with StatusCode set to *UNKNOWN_PASSWORD_IDENTIFIER*
>
> §12.4.8.6.3

However, this requirement infers the following safety property:

$$AG(\neg(\text{state}(pi) = \mathsf{Nothing} \wedge \text{event}(pi) \neq \mathsf{Del}))$$

indeed, in case of a failure, PIs (pi) (in the state Nothing) should always request to be deallocated by the PP by sending the termination *event* Del. Failing, the property returns a trace where the PI remains stuck in the Nothing state. As shown by the requirement, the Del event is missing, and this leads the PI to become unresponsive and the PP can no longer remove it.

This attack, in turn, leads to the violation of additional safety properties in the PP model. More precisely, the standard in a note states:

> NOTE—A protocol instance in Nothing state will never receive an SAE Confirm message due to the state machine behaviour of the parent process
>
> §12.4.8.6.3

However, by analysing the behaviour of the PI state machine when it is stalled, we discovered that the PP can send the `CON` event to the PI that is waiting in the Nothing state (this is because the PP only checks the presence of a PI with the correct MAC but does not check the PI state). The PI does not handle the `CON` in the Nothing state, and this could further aggravate the situation as unhandled severe exceptions could be triggered.

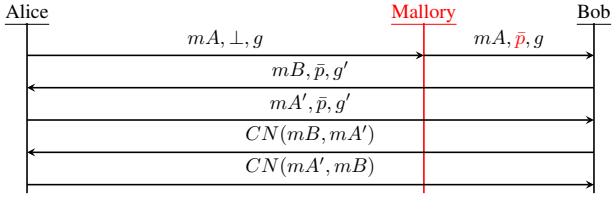
To ensure that the standard is safeguarded against possible future regressions, it would be best to include two *patches*: one for PP and one for PI. The *first patch* should require PP to not only check if the MAC associated with the one in the confirm message is present in its database but also verify the PI's state. The *second patch* (which, at the current state of the specification, patches both) would involve sending the Del event to PP, in addition to the frame with StatusCode set to `UNKNOWN_PASSWORD_IDENTIFIER` when the PI receives a message containing a password identifier not present in the table.

6.3. Unified results: Secure specification for password identifiers

At the Communication level, two peers are not forbidden to choose different password identifiers, but when the two password identifiers are different, the protocol will fail without exchanging a common key (correctness violation). Surprisingly, the analysis of the state machine highlighted that two peers are *not* allowed to choose different password identifiers (failure), contradicting the protocol specification. We discovered that the only states where a renewal of the commit message could have happened was the Commited. The main case of interest turned out to be the agreement on groups, as, if two peers initiated the protocol with different but supported groups, then one group would be selected by both on the basis of the natural ordering of their MAC address: in detail, the one with the lowest MAC has to recommit and confirm, and the other has to ignore and wait for another commit message.

We leveraged this result back to our model at the Communication level to include group agreement according to the specification, as well as our *patch* to handle password identifiers, see Sec. 6.2. From the Communication perspective, we end up finding a *false* attack that seems to downgrade password identifiers.

Figure 7. Attack on password identifier WPA3-SAE protocol in IEEE 802.11:2020 at the Communication level, that reveals to be a *false* attack when we analyse the Device level.



The Communication level model confirms the existence of this attack, allowing two peers to start with different groups and password identifiers. However, the state machine actually implements a check on this, disallowing the start of the third message in Fig. 7.

This demonstrates that the flaw handled at the Device level (without a justification) is covering for a vulnerability affecting the Communication level, supporting the high number of misalignments in our Errata. This is to exemplify that our unified approach allowed our formal verification to enjoy a deeper security analysis than traditional single-model verification. We argue that the correct design of the protocol (which is what our patch accounts for) should supersede the patching of the incorrect design at a different level. Additionally, to the patch we proposed for handing password identifiers; the deeper analysis here sketched also suggests that two other important points have to be added to the Communication level specifications: i) despite both peers being allowed to start the protocol, when one receives before having sent its first message, it should consider itself a Receiver and act accordingly; and ii) a Receiver should

always accept a supported group, never offer a different group, and terminate the protocol in a failure state, if an unsupported group is offered according to the expected rejection list of groups. *Our verification shows that with such specifications, the above attack will not occur.*

7. Implementing a Secure WPA3-SAE Protocol

Using the latest version of hostapd (https://w1.fi/ - hostapd 2.10), a widely used open source network driver for linux, as reference, we investigate it's security and adherance to the standard.

First begin with the independent findings from Section 6.1. Immediately we see that the code actually disallows for the s_L and E_l to be the same through a check.

```
1 if (sae->state == SAE_ACCEPTED &&
2   sae->peer_commit_scalar_accepted &&
3   crypto_bignum_cmp(sae->
4    peer_commit_scalar_accepted,peer_scalar) == 0) {
5                   ...
6                   crypto_bignum_deinit(peer_scalar, 0);
7                   return
                        WLAN_STATUS_UNSPECIFIED_FAILURE;
8   }
```

Interestingly, this is noted as a fix introduced in IEEE Std 802.11-2012, however this is removed from the standard for the 2020 version. Perhaps as an outcome of previous vulnerabilities which only focused on implementations and not the standard.

Likewise as per our suggested changes, indeed hostapd has error handling in the committed phase of PI, thus not leading to the deadlock described in 6.1, we note our proposed fix would allow for more informative failures, whilst this equates to a hard reset no matter which error (as the procedure for handling this was not previously specified in the standard this is understandable).

```
1 case SAE_COMMITTED:
2   sae_clear_retransmit_timer(hapd, sta);
3   if (auth_transaction == 1) {
4     if (sae_process_commit(sta->sae) < 0)
5       return WLAN_STATUS_UNSPECIFIED_FAILURE;
6   ...
```

In the joint findings, there is indeed once again a deviation from the specification, and an ambiguous decision. We see that in the case of `WPA_EVENT_SAE_UNKNOWN_PASSWORD_IDENTIFIER`, there is further actions than those specified in the standard.

```
1 if (resp ==
        WLAN_STATUS_UNKNOWN_PASSWORD_IDENTIFIER) {
2   wpa_msg(hapd->msg_ctx, MSG_INFO,
        WPA_EVENT_SAE_UNKNOWN_PASSWORD_IDENTIFIER
        , MACSTR, MAC2STR(sta->addr));
3   sae_clear_retransmit_timer(hapd, sta);
4   sae_set_state(sta, SAE_NOTHING,
5       "Unknown_Password_Identifier");
6   goto remove_sta;
7 }
```

We see here a goto call to `remove_sta`, which is highlighted below.

```
1 remove_sta:
2         if (auth_transaction == 1)
3                 success_status = sae_status_success(hapd,
                      status_code);
4         else
5                 success_status = status_code ==
                      WLAN_STATUS_SUCCESS;
6         if (!sta_removed && sta->added_unassoc &&
7           (resp != WLAN_STATUS_SUCCESS || !success_status)
                ) {
8                 hostapd_drv_sta_remove(hapd, sta->addr);
9                 sta->added_unassoc = 0;
10        }
11        wpabuf_free(data);
12 }
```

The `remove_sta` section checks the status of an authentication attempt. If unsuccessful (based on `status_code` or other conditions), it removes the unassociated STA from the driver and frees allocated resources. This is in theory a somewhat correct step, however it brute forces the problem once again (and it is not compliant with how the separation of agents should work), rather than the more precise solution proposed in our patches from Section. 6.2. Finally, the joint finding that the check is done at a different layer is confirmed in this implementation as well, and suffers from the same downsides.

What we see from this analysis is that not only are there several deviations from the standard in the largest open-source implementation of WPA3, but furthermore the ambiguity of the specification led to suboptimal fixes, which could introduce future errors (or even be removed by incautious maintainers as they are deviations from the standard rather than being required by it).

8. Conclusions

We introduced a unified methodology based on the integrated use of two different verification formalisms and techniques to address a critical analysis of WPA3's security framework. Our analysis approach combines the traditional modelling technique for protocol analysis, mechanized in ProVerif, and state machine formalism, mechanized in ASM-META. This integration leverages the strengths of each approach to achieve the verification goals of each technique and allows for discovering vulnerabilities that might go undetected by classical single-level verification processes. Vulnerabilities detected by the two formalisms sometimes overlap, in some cases offer disjointed results, and in other cases, the outcomes from one of the two formalisms refined our analysis of the other, with deeper security insights and further attacks detected. This unified approach allowed us to discover security issues, and strongly support patches that we proposed, in the form of over 20 errata, to the IEEE 802.11 Working Group. Most of our patches have been accepted and will appear in the next revision of the Wi-Fi Standard.

Acknowledgments

The authors would like to thank Anonymous Colleagues for their valuable input into the writing of this work.

References

[1] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1773–1788.

[2] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 435–450.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani *et al.*, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005. Proceedings 17*. Springer, 2005, pp. 281–285.

[4] L. Arnaboldi and H. Tschofenig, "A formal model for delegated authorization of IoT devices using ACE-OAuth," in *OAuth Security Workshop*, 2019.

[5] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz, and T. Würtele, "DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 523–542.

[6] C. Braghin, M. Lilli, and E. Riccobene, "A model-based approach for vulnerability analysis of IoT security protocols: The Z-Wave case study," *Computers & Security*, vol. 127, p. 103037, 2023.

[7] R. Metere and C. Dong, "Automated cryptographic analysis of the pedersen commitment scheme," in *Computer Network Security: 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings 7*. Springer, 2017, pp. 275–287.

[8] K. Ye, R. Metere, and P. Yadav, "User-Guided Verification of Security Protocols via Sound Animation," *arXiv preprint arXiv:2410.00676*, 2024.

[9] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd," in *IEEE Symposium on Security & Privacy (SP)*. IEEE, 2020.

[10] D. Basin, C. Cremers, J. Dreier, and R. Sasse, "Tamarin: Verification of Large-Scale, Real-World, Cryptographic Protocols," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 22–31, 2022.

[11] A. Armando, D. Basin, J. Cuellar, M. Rusinowitch, and L. Viganò, "AVISPA: automated validation of internet security protocols and applications," *ERCIM News*, vol. 64, no. January, 2006.

[12] B. Blanchet, V. Cheval, and V. Cortier, "ProVerif with lemmas, induction, fast subsumption, and much more," in *IEEE Symposium on Security and Privacy (S&P'22)*. San Francisco, CA: IEEE Computer Society, May 2022, pp. 205–222.

[13] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[14] L. Arquint, M. Schwerhoff, V. Mehta, and P. Müller, "A generic methodology for the modular verification of security protocol implementations," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1377–1391.

[15] K. Morio, D. Jackson, M. Vassena, and R. Künnemann, "Modular black-box runtime verification of security protocols," *PLAS 2020*, 2020.

[16] L. Arnaboldi and R. Metere, "Poster: Towards a data centric approach for the design and verification of cryptographic protocols," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2585–2587.

[17] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, "Implementing TLS with verified cryptographic security," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 445–459.

[18] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[19] I. C. S. L. S. Committee *et al.*, "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11ˆ*, 2007.

[20] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," Tech. Rep., 2021. [Online]. Available: https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf

[21] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[22] E. Börger and A. Raschke, *Modeling Companion for Software Practitioners*. Springer, 2018.

[23] A. Bombarda, S. Bonfanti, A. Gargantini, E. Riccobene, and P. Scandurra, "ASMETA Tool Set for Rigorous System Design," in *International Symposium on Formal Methods*. Springer, 2024, pp. 492–517.

[24] A. Carioni, A. Gargantini, E. Riccobene, and P. Scandurra, "A scenario-based validation language for ASMs," in *Abstract State Machines, B and Z: First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings 1*. Springer, 2008, pp. 71–84.

[25] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11*. Springer, 1999, pp. 495–499.

[26] F. Hao, R. Metere, S. F. Shahandashti, and C. Dong, "Analyzing and Patching SPEKE in ISO/IEC," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2844–2855, 2018.

[27] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 483–502.

[28] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.