Model Checking the IKEv2 Protocol Using Spin

Tristan Ninet*†, Axel Legay‡, Romaric Maillard†, Louis-Marie Traonouez* and Olivier Zendra*
* Inria {tristan.ninet,louis-marie.traonouez,olivier.zendra}@inria.fr
† Thales SIX GTS France romaric.maillard@thalesgroup.com
‡ U.C. Louvain axel.legay@uclouvain.be

***Abstract*—Previous analyses of IKEv2 concluded that the protocol was suffering from two authentication vulnerabilities: the penultimate authentication flaw and a vulnerability that leads to a reflection attack. In this paper, we analyze the IKEv2 protocol specification using the Spin model checker. To do so, we extend and improve an existing modeling method that allows analyzing security protocols using Spin. For completeness, we indicate each abstraction we make when writing the model. As a result, we confirm the penultimate authentication flaw and show that the reflection attack is actually not applicable.**

***Index Terms*—Spin, model checking, IKEv2, authenticated key-exchange, security protocols**

I. Introduction

Internet Key Exchange version 2 (IKEv2) is the authenticated key-exchange protocol used in the Internet Protocol security architecture (IPsec). A security protocol such as IKEv2 can suffer from two types of vulnerabilities: specification vulnerabilities and implementation vulnerabilities.

An appropriate approach to avoid implementation vulnerabilities is to use modern automated testing techniques like fuzzing. As a matter of fact, the developers of the strongSwan IKEv2 implementation have recently announced [1] that part of their code base is now fuzzed using Google's OSS-Fuzz [2] infrastructure. Fuzzing is an active research field in which a lot of progress has been made, e.g. by embracing additional techniques such as symbolic execution.

A specification vulnerability is inherent to the protocol itself and cannot be fixed by any change in the implementation. An efficient way to find specification vulnerabilities is to use automated techniques, such as model checking. Model checking allows to detect specification flaws in early stages of the development process, which in turn reduces total cost of solving the flaws. Furthermore, model checking is an exhaustive technique: it can formally prove that a protocol specification model meets its goals. Finally, security protocols are often too complex to rely only upon human understanding: IKEv2 is made of sixteen different payloads and even more substructures and fields. IKEv2 contains a mechanism of rekeying, which negotiates the secret keys periodically. It is designed so as to work even in the presence of Network Address Translation (NAT) between the peers. IKEv2 specifies not less than twenty-nine types of notification and error messages. Facing such a complexity, it seems sound to use an automated process to verify IKEv2.

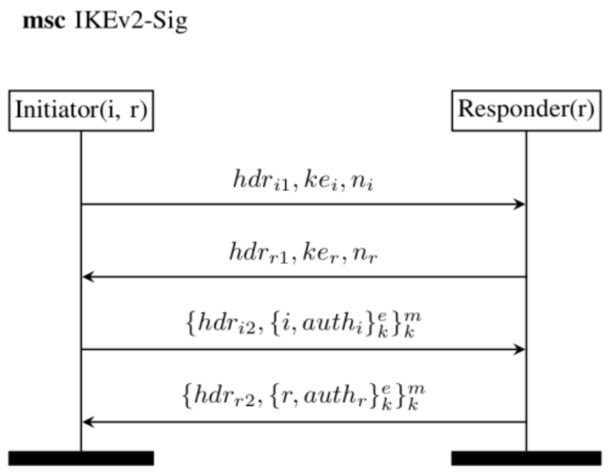
Model checking has been used, to our knowledge, twice to analyze IKEv1 [3], [4] and three times to analyze IKEv2 [4]–[6]. Tools that were used are NRL [7], OFMC [8], Scyther [9] and DH-ProVerif [6]. It revealed two authentication vulnerabilities that were found in both IKEv1 and IKEv2: the penultimate authentication flaw and the reflection attack.

In this paper, we start by formally describing IKEv2's payloads in Section II. We further give some background on model checking security protocols, previous analyses of IKEv2, and the Spin model checker in Section III. We then analyze IKEv2 using Spin. To our knowledge it is the first time that Spin is used to analyze such a complex protocol. To do so, we extend and improve an existing method [10] for modeling protocols in Promela (Spin's modeling language). We explain our modeling method in section IV, and provide the source code for reusability. We observe that, even with our modeling method, several abstractions are inevitable when writing a model of a security protocol in Promela. Other tools such as ProVerif [11] and Tamarin [12] offer more realistic models.

Our results, detailed in Section V, confirm the penultimate authentication flaw, but show that the reflection attack has no practical existence against IKEv2. The model of [4], which reported the vulnerability, was missing some payloads that actually prevent the attack. Furthermore, our analysis shows that IKEv2-PSK and IKE-Child satisfies non-injective agreement in our adversary model.

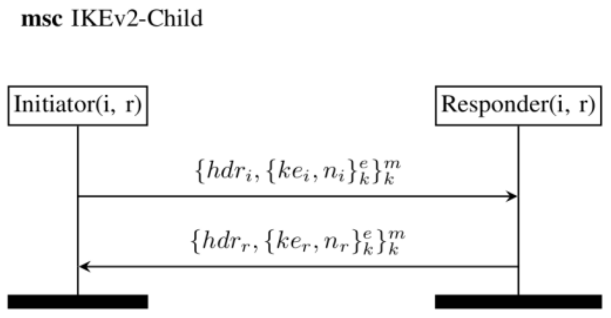
II. The IKEv2 protocol

The IKEv2 specification [13] is managed by the Internet Engineering Task Force (IETF). The goal of IKEv2 is to allow two peers to dynamically negotiate cryptographic algorithms and material in order to set up an IPsec [14] security association (SA). A security association is a set of security parameters and keys which enables two peers to exchange protected traffic. IKEv2 consists of three main exchanges: IKE_SA_INIT, IKE_AUTH and CREATE_CHILD_SA. During the IKE_SA_INIT exchange the two peers negotiate cryptographic algorithms and run a Diffie-Hellman protocol [15] to generate a shared secret. Keying material derived from this secret will be used with the algorithms agreed upon to encrypt subsequent IKEv2 messages. The result of an IKE_SA_INIT is called an IKE SA. Once the IKE SA is initiated, the two peers perform mutual authentication using the IKE_AUTH exchange to deter Man-in-the-Middle attacks. This authentication can be based on either pre-shared keys or digital certificates. This IKE_AUTH exchange is also used to establish an initial IPsec SA. Subsequent IPsec SAs and IKE SAs will be created through the CREATE_CHILD_SA



$$
\begin{array}{ll}
ke_i = g^{x_i} & hdr_{il} = (rf_{il}, if_{il}, mid_{il}) \\
ke_r = g^{x_r} & hdr_{rl} = (rf_{rl}, if_{rl}, mid_{rl}) \\
keymat = k & auth_r = \{ke_r, n_r, n_i, k, r\}^s_{prk(r)} \\
k = (ke_i)^{x_r} = (ke_r)^{x_i} & auth_i = \{ke_i, n_i, n_r, k, i\}^s_{prk(i)}
\end{array}
$$

Fig. 1. The IKEv2-Sig protocol. This is the first phase of IKEv2, using signature authentication method.



msc IKEv2-Child
Initiator(i, r)
Responder(i, r)
$\{hdr_i, \{ke_i, n_i\}^e_k\}^m_k$
$\{hdr_r, \{ke_r, n_r\}^e_k\}^m_k$

$$
\begin{array}{l}
hdr_i = (rf_i, if_i, mid_i) \\
hdr_r = (rf_r, if_r, mid_r) \\
keymat = (ke_i)^{x_r} = (ke_r)^{x_i} \\
ke_i = g^{x_i} \\
ke_r = g^{x_r}
\end{array}
$$

Fig. 2. The IKEv2-Child protocol. This is the second phase of IKEv2.

exchange (possibly replacing existing SAs for the purpose of rekeying).

IKEv2 aims to guarantee mostly two security properties. First, that the keying material generated by the IKE_SA_INIT and CREATE_CHILD_SA exchanges is secret, i.e. is only known to the two parties involved. Second, that the parties involved are mutually authenticated: each party must prove that it really has the identity it pretends to have. In this paper, we use model checking to verify that IKEv2 actually satisfies these two properties.

Syntax	Semantics
$prk(a)$	Private key of a
$pbk(a)$	Public key of a
$psk(a, b)$	Pre-shared key of a and b
g	Diffie-Hellman generator
hdr_{al}	lth IKEv2 header sent by a
ke_a	Key-exchange payload sent by a
n_a	Nonce payload sent by a
$auth_a$	Authentication payload sent by a
x_a	Diffie-Hellman exponent of a, which is not sent
rf_{al}	lth Response flag field sent by a
if_{al}	lth Initiator flag field sent by a
mid_{al}	lth message ID field sent by a
$\{msg\}^e_k$	Symmetric encryption of msg using key k
$\{msg\}^s_k$	Digital signature of msg using k if k is a private key. Keyed hash of msg using k if k is a pre-shared key
$\{msg\}^m_k$	Message msg in plain text but integrity protected by a MAC using key k
q^r	Modular exponentiation (exponentiation in a finite field) of q by r, where $q, r \in \mathbb{N}^*$
$Initiator(i, r)$	Agent i is taking the role of initiator and wants to perform the protocol with agent r
$Responder(r)$	Agent r is taking the role of responder and can perform the protocol with whatever agent correctly authenticates itself and is trusted by r
$Responder(i, r)$	Agent r is taking the role of responder and can only perform the protocol with agent i
k (in IKEv2-Child)	Pre-shared key for IKEv2-Child

Fig. 3. Our syntax for payloads and fields of IKEv2's subprotocols. In this syntax, a denotes an agent, and $l \in \mathbb{N}^*$. We write hdr_a, rf_a, if_a and mid_a when the subprotocol contains only one exchange

To simplify the model checking process, we focus on specific parts of IKEv2. These parts constitute protocols on their own, so we call them *subprotocols*. We define the following ones:

IKEv2-Sig consists of one IKE_SA_INIT exchange and one IKE_AUTH exchange. It uses digital signature authentication.

IKEv2-PSK consists of one IKE_SA_INIT exchange and one IKE_AUTH exchange. It uses pre-shared key authentication.

IKEv2-Child consists of one CREATE_CHILD_SA exchange, where key-exchange payloads are included (hence where a Diffie-Hellman protocol is run).

Figures 1 and 2 show the message sequence charts (MSC) of IKEv2-Sig and IKEv2-Child. The IKEv2-PSK MSC can be obtained from the IKEv2-PSK by replacing $prk(i)$ and $prk(r)$ with $psk(i, r)$. Figure 3 shows the syntax we use in these MSCs.

The Response flag of a message is set to 1 when the message is a response, and the Initiator flag is set to 1 when the message is sent by the IKE SA original initiator. The message ID is an integer that starts with value 0 and is incremented at every new exchange. Its role is to prevent replay attacks, as well as to detect retransmissions and lost messages.

Ideally, we would like to write models that represent the exact behaviour of IKEv2 as defined in its RFC. However, IKEv2 is far too complex to be fully modeled. For this reason, we choose to model only a subset of IKEv2. In other words, we perform some abstractions when writing our models.

We say that an abstraction is sound when the following holds: if our model satisfies some property, then removing the abstraction yields a model that satisfies the same property. We say that an abstraction is correct when the following holds: our model satisfies some property if and only if removing the abstraction yields a model that satisfies the same property.

It is not always obvious whether an abstraction is correct or not. This is the reason why e.g. [4] finds an attack that does not exist: the reflection attack. We discuss the reflection attack in Section V.

As a first class of abstraction, we do not keep all IKEv2 payloads and fields in our models. For example, we do not include the traffic selector (TS) payloads. TS payloads are used to specify which IP addresses will be allowed to communicate through the resulting Child SA. TS payloads are not used in any cryptographic operation. They neither play a role in key generation nor in authentication. Therefore, we can reasonably assume that this abstraction is sound.

Other abstractions we make concern the way cryptographic values are computed by parties.

In our models, all values *SKEYSEED*, *SK_ai*, *SK_ar*, *SK_ei*, *SK_er*, *SK_pi*, *SK_pr* and *SK_d* of the IKEv2 RFC [13] are represented by a single value k. *KEYMAT* is represented by $keymat$. $keymat$ is therefore the term of which we want to verify the secrecy.

Because key derivation algorithms of IKEv2 are public, an attacker who learns *SKEYSEED* can also learn all SK_* keys. However, the opposite is not true because key derivation in IKEv2 uses a hash function. In our model, compromising one of these values is equivalent to compromising all other values. Since our abstraction of key derivation only grants some additional capability to the attacker, we can reasonably assume that this abstraction is sound.

III. Background

A. Model checking security protocols

Formal verification is the act of proving or disproving that a system satisfies some property using a mathematically based technique. Model checking is a formal verification technique in which we represent the system by a model, whose semantics is a transition system, and explore systematically and exhaustively all its states and transitions in order to prove that it satisfies the property. The model checker takes as input the system model, as well as a property over the model state variables, and either returns *Yes*, returns *No*, or does not give any response (e.g. by not terminating). When they return no and when they can, some model checkers also give a counter-example, i.e. an execution trace of the model that contradicts the property. The principles of model checking are explained in great details in [18], [19].

We want a security protocol to be secure even in the presence of an adversary that can intercept, drop, learn from and build messages. To formalize this, Dolev and Yao first defined [20] what is now called the symbolic model, or Dolev-Yao model. In this model, messages are abstracted away as entities, cryptography is supposed to be flawless and the intruder has full control over the network. The adversary's knowledge and the advancement of some agents in their execution of the protocol can be seen together as constituting a symbolic state. The actions "an agent sends a message", "an agent receives a message", "the adversary builds a message and sends it", etc., can be seen as actions modifying the state. Such a model thus lends itself well to model checking techniques.

Therefore, in our case, the system mentioned earlier is a protocol specification, played in some adversary model (capabilities given to the intruder), and the properties are security properties, such as secrecy and authentication. A thorough state-of-the-art of model checking security protocols is depicted in [21].

B. Related work

In 1999, Meadows finds two authentication weaknesses in IKEv1 [3], using the NRL protocol analyzer. The first one is a reflection attack, and the second one is called the penultimate authentication flaw. We explain these later.

In 2003, IKEv2 is formally verified in the context of the AVISPA project [5]. The authors find that IKEv2 also suffers from the penultimate authentication flaw. However, they say that it cannot be exploited for further purposes. They propose a counter-measure anyway: key confirmation.

In 2009, Kusters and Truderung use their tool DH-ProVerif to verify IKEv2 [6]. Their analysis seems to confirm the penultimate authentication flaw.

In 2010, Cremers performs an extensive analysis of IKEv2 [4] using the Scyther tool. He confirms that IKEv2 suffers from the penultimate authentication flaw and, like in the AVISPA project, concludes that this vulnerability is harmless. In addition, [4] finds that the reflection attack that was noted for IKEv1 is also possible on IKEv2.

C. The Spin model checker

Spin [22] is a general-purpose explicit-state model checker. It takes Promela [17] as input language and was designed to check LTL properties on asynchronous process systems. Spin translates processes into finite-state automata (hence the adjective explicit-state), performs an interleaving product on them and searches the resulting state space for a property violation. Since a protocol is an asynchronous process system, and since all the properties we want to verify are safety properties (which are LTL properties), Spin can be used for protocol verification. However, it lacks native support for cryptographic primitives and for an adversary model. To solve this, one can use the method introduced in [10]. We describe this method later in Section IV-C.

We observe that, even with the method of [10] and our improvements to this method, security proofs obtained using Spin are weaker than security proofs obtained using other tools that are specialized in security protocol model checking. Such tools include ProVerif [11], Scyther [9] and Tamarin [12]. We discuss that matter in Section VI.

IV. Model checking IKEv2 using Spin

A. Choosing our adversary model

The property we verify strongly depends on the capabilities given to the intruder, during verification. Consequently, [23] proposes to split each security property into an adversary model and an *atomic property*. We follow the same principle in our model.

[23] translates several adversary models from the literature into their own formalism. We implement one of them in our model: the internal Dolev-Yao model (*AdvINT*). In AdvINT, cryptography is supposed to be perfect, i.e. the intruder can only decrypt a message if it possesses the decryption key. In addition, the intruder has full control over the network: it learns from all messages that are sent and can inject its own forged messages into the network.

Furthermore, the intruder has the *LKRothers* capability, which allows him to compromise, at the beginning of the model execution, the long-term keys of any agent that is not mentioned in the property we are verifying.

B. Choosing our properties

Ideally, we would like to verify the exact properties that the protocol claims to guarantee. However, security properties can be quite vague. In particular, the "mutual authentication" mentioned in [13] is an unclear notion. For this reason, researchers have split authentication into several well-defined properties [24], [25]. We verify the following atomic properties:

Secrecy of $keymat$ This property states that whenever an agent has completed the protocol, the term $keymat$ that it computes will never be known to the intruder.

Aliveness This property states that whenever an agent A has completed the protocol, apparently with an agent B, then B has previously been running the protocol.

Weak agreement This property states that whenever an agent A has completed the protocol, apparently with an agent B, then B has previously been running the protocol, apparently with A.

Non-injective agreement This property states that whenever an agent A has completed the protocol, apparently with an agent B, then B has previously been running the protocol, endorsing the correct role, apparently with A, and A and B agree on some terms.

Aliveness, weak agreement and non-injective agreeement are authentication properties and were first defined by Lowe in [24]. For the initiator, "apparently with B" means that the i payload it received in the IKE_AUTH response equals B. For the responder, "apparently with A" means that the r payload it received in the IKE_AUTH request equals A. Note that this makes weak agreement different from aliveness.

"B has previously been running the protocol" means that B has at least sent its last message. Obviously, A cannot have any stronger guarantee: if B's last event is a "receive", then the protocol cannot prove to A that this event was triggered.

We consider our authentication properties satisfied if they are satisfied whatever role A is endorsing, i.e. if they are satisfied for both the initiator and the responder.

Note that non-injective agreement implies weak agreement, which in turn implies aliveness. There are stronger authentication properties that we could verify. The *agreement* property, for example, adds to non-injective agreement the condition that if the partner has completed the protocol and computed the same agreement terms more times than the actor (this prevents replay attacks).

C. The existing modeling method

A problem we face in IKEv2 modeling is that Promela was not designed to model security protocols, but rather more general asynchronous process systems. For example, it does not provide a simple way to model encryption. We thus use the method explained in [10].

1) Modeling the protocol: In this method, the network is modeled by a single channel. The channel size is set to the maximum of all protocol message sizes. The size of a message is defined as the number of payloads it contains, plus the number of times an encryption appears in the payloads.

The method defines a set of *mtype* (a Promela type of variable) constants, which we call *names*, describing all possible values that a payload of the protocol may take.

Each role is modeled by a process taking agent names as input. These names represent the identity of the agent running the role and the intended partner of the agent (when there is one). To model encryption, i.e. to model an agent sending $\{x\}_k^e$, [10] writes $Chan!k, x$. This is why each encryption defined by -the protocol increases the size of the channel.

Finally, processes are instantiated. We instantiate two sessions: agent A taking the role of initiator and non-deterministically intending to speak with B or C, and agent B taking the role of responder.

2) Modeling the adversary: The intruder is modeled by a process (just like the initiator and responder roles), that non-deterministically chooses between receiving from the channel, or forging a new message and sending it.

The intruder's knowledge is modeled by some boolean vector called $Knows$ indexed by all names. When a $Knows[name]$ is set to 1, it means that the intruder knows $name$. [10] further uses some local variables in the intruder process so that the intruder can store exactly one message. This represents the ability of the attacker to store some encrypted payload for later decryption in case he currently cannot decrypt the payload.

When the intruder chooses to receive a message, the intruder updates his knowledge using a function called $Addtoknowledge$. If there is an encrypted payload k, x, the intruder can learn x if and only if he possesses k^{-1}. Then the intruder can choose (or not) to update his memory by storing the message, thus forgetting any previously stored one. Finally, the intruder can choose (or not) to forward the message or do nothing. The latter case represents the dropping of a message.

To create a new message, the intruder calls a function called $Randmessage$, which non-deterministically generates a message using names known to the intruder. The intruder then uses a macro called $Isvalidmessage$, which checks the consistency of the generated message with the intruder knowledge.

In the main process, we set the intruder's initial knowledge and run the intruder process. [10] gives Charlie's long-term keys to the intruder knowledge, thereby seeking to implement the AdvINT adversary model.

3) Modeling the properties: Each security property is expressed as a boolean condition over state variables. The boolean condition is called *invariant*. If the invariant becomes false during execution, then the corresponding property is violated. Each invariant is checked at each exection step via a dedicated process.

Values of state variables are set during protocol execution. The goal of these variables is to keep track of: (a) where each agent is at in its protocol execution, (b) to whom the agent believes it is talking, and (c) what value the agent has computed for $keymat$.

[10] implements secrecy and some form of authentication. His implementation of authentication can be seen as weak agreement with the additional condition that the peer has the correct role.

D. Extending the existing method

We improve the method of [10] in order to fit the model described in Sections IV-A and IV-B.

1) Extending protocol modeling: Firstly, we propose some way to model the Diffie-Hellman protocol. The Diffie-Hellman protocol can be written as such:

$$i \rightarrow r : g^{x_i}$$
$$r \rightarrow i : g^{x_r}$$

Where i and r are agents, g is a constant and x_i and x_r are nonces. At the end of the protocol, i and r both compute the value $g^{x_i * x_r}$.

To model the Diffie-Hellman protocol in Promela, we add name G representing g. We add names XA, XB and XC representing values x_a, x_b and x_c. Each agent executes the protocol at most once in an execution trace of our model, so each agent will need at most one value of type X to send.

We add names KEA, KEB and KEC representing values g^{x_a}, g^{x_b} and g^{x_c}. We add names KAA, KAB, KAC, KBB, KBC and KCC representing values $g^{x_a * x_a}$, $g^{x_a * x_b}$, $g^{x_a * x_c}$, $g^{x_b * x_b}$, $g^{x_b * x_c}$ and $g^{x_c * x_c}$.

We create a function called $Dhexp$, which maps represents modular exponentiation by mapping a couple (ke, x) to a name k.

Finally, modeling the Diffie-Hellman protocol requires adding a deduction step in function $Addtoknowledge$. Indeed, when the intruder learns a new name, we now need to check whether he can deduce some key by modular exponentiation.

To analyze IKEv2, we further need to model signature and integrity protection. To do so, we naturally extend the encryption modeling method of [10]. To model $\{x\}_k^s$, we use: k, x, and the intruder can never learn x, even if he possesses k^{-1}. To model $\{x\}_k^m$, we use: k, x, and the intruder can always learn x, even if he does not possess k^{-1}.

2) Extending adversary modeling: [10] proposes several definitions of $RandMessage$, but none of them were efficient enough for our analysis. In this Section, we propose a more efficient way to model message creation by the intruder.

Consider some message that the intruder sends to some agent. In order to be faithful to AdvINT, we must check that the intruder's knowledge allows him to generate the message. Then for each payload, we check that the type is right. [10] checks consistency with intruder knowledge in his $Isvalidmessage$ function, and performs type checking in role processes.

In order to avoid state explosion, we propose to directly check type and consistency with intruder knowledge in $Randmessage$. More concretely, our $Randmessage$ directly chooses for each payload a value among names whose type fits the payload. When using this method, there is no need for some $Isvalidmessage$ macro since $Randmessage$ always generates a valid message. Our method greatly limits path explosion and allows our analysis to terminate.

3) Extending properties modeling: In its secrecy invariant, [10] only checks that some constant is not known to the intruder. This constant is sent by one of the role in the Denning-Sacco protocol, which he uses as an example.

We cannot proceed this way for IKEv2. In order to model secrecy in IKEv2, we need to check that any name that an agent considers as its secrecy term is not known to the intruder.

We then introduce variables sta and stb to store what agents A and B consider as their secrecy term, and use these variables in our secrecy implementation.

To implement aliveness, we need two new boolean variables, namely $arunning$ and $brunning$. We set these variables in our roles when A (resp. B) has been running the protocol.

Implementation of weak agreement is straightforward and resembles the authentication implementation of [10], except that we do not require that the partner endorse the correct role.

To implement non-injective agreement, we further need to store the names on which each agent agrees when "running" and when "committing". We therefore define variables $atra$, $atrb$, $atca$ and $atcb$, where e.g. $arta$ means "agreement term of a at running step".

V. Analysis results

Our Promela code and the exact Spin commands we used are available at [26]. We present our analysis results in table 4. Allowing only two sessions in a trace, very few amount of time and memory was necessary to perform the verifications. For example, only 3s and 128 MB of memory were necessary to prove aliveness on IKEv2-Sig in AdvINT. Note that we used the bitstate hashing optimisation. Our analysis yields two notable results. Firstly, it proves that IKEv2-PSK and IKE-Child satisfies non-injective agreement in our adversary model. Secondly, it refutes the reflection attack that was found by previous analyses.

[4] claims that IKEv2-Child is vulnerable to a reflection attack against the initiator. In this attack, the intruder replays the initiator's CREATE_CHILD_SA request to itself. The initiator then responds to this request, and the intruder replays this response to the initiator. This would result in a violation of aliveness, since the initiator would have thought having set up a connection with an other agent, when in fact it would have set it up with itself, the other agent not even being alive.

However, [4] does not include the two *Initiator* and *Response* flags of the IKEv2 header in his model. Their role is explained in Section II. By adding these flags, our analysis shows that IKEv2-Child satisfies aliveness, weak agreement and non-injective agreement. Indeed, because of these flags, during a reflection attack, the initiator will notice that the request it receives comes from the original initiator (which is itself). He will thus refuse to answer it. Furthermore, the flags are integrity-protected: the RFC of IKEv2 says (and we found through experiment that it was the case in the strongSwan implementation) that the "Integrity Checksum Data" field of the encrypted payload is "the cryptographic checksum of the entire message starting with the Fixed IKE header through the Pad Length". The intruder thus cannot successfully change these payloads without knowing key k. Since secrecy of k is satisfied, the reflection attack is not possible.

[4] already pointed out the obvious defense against the reflection attack: "breaking the symmetry of the messages, e. g., by including distinct constants and checking their presence". We have shown that this defense is already in place in the protocol. Furthermore, the defense involves more than simply including distinct constants: these constants need to be integrity protected.

Our analysis also confirms that IKEv2-Sig does not satisfy weak agreement. This vulnerability is called the penultimate authentication flaw and was already found in previous analyses. This vulnerability is not a full violation of the intuitive definition of authentication, because there is no actual impersonation and secrecy is still satisfied. However, it has been shown in [27] that the penultimate authentication flaw makes it possible to perform a Denial-of-Service attack against IKEv2. Therefore it is important that IKEv2 satisfies weak agreement. We refer the reader to [27] for two possible modifications of IKEv2 which both deter the penultimate authentication flaw.

VI. Discussion

Even with the method [10], the nature of Promela inherently adds a layer of unsound abstractions to the modeling process.

Because Spin is bounded and because of the constraints in time and memory, we allow a maximum of two sessions in an execution trace and use three agents (A, B and C). This an unsound abstraction. Nevertheless, such a model can capture a large class of attacks.

A memory of exactly one message is given to the intruder. This is an unsound abstraction of the adversary model. Unfortunately, increasing this memory quickly leads to a path explosion.

The intruder of AdvINT can encrypt some value an arbitrary number of times before sending it. Our Promela model does not allow that, which constitutes an unsound abstraction.

As explained in Section IV-D2, we have moved type checking to the adversary process. However, to be faithful to AdvINT, type checking should be performed in role processes. Moving type checking to adversary thus is an abstraction. However, we consider this abstraction as correct because receiving a malformed message direclty causes the execution path to stop anyway.

Due to our unsound abstractions, the model of [4] captures a larger class of attacks than ours. Therefore, the analysis of IKEv2 performed in [4] using Scyther renders more meaningful results than ours regarding the proof of secrecy, aliveness and weak agreement on IKEv2-Sig and IKEv2-PSK. Our contribution regarding the security of IKEv2 lays in refuting the reflection attack of [4], in proving that IKEv2-Child satisfies secrecy, aliveness and weak agreement in our adversary model, and in proving that IKEv2-PSK and IKEv2-Child satisfy non-injective agreement in our adversary model.

VII. Conclusion and future work

In this paper, we have performed a formal analysis of the IKEv2 specification using the Spin model checker. To do so we extended the method of [10] with (among others) a model of Diffie-Hellman exponentiation, an implementation

Property	IKEv2-Sig	IKEv2-PSK	IKEv2-Child
Secrecy	✓	✓	✓
Aliveness	✓	✓	✓
Weak agreement	✗	✓	✓
Non-injective agreement	✗	✓	✓

Fig. 4. Analysis of IKEv2 using Spin. We write ✓when a subprotocol satisfies some property in our adversary model, and ✗when it does not. A subprotocol satisfies a property if and only if the property is satisfied for both the initiator and the responder. Our analysis refutes the reflection attack that was found by previous analyses and proves that IKEv2-PSK and IKE-Child satisfies non-injective agreement in our adversary model.

of authentication properties following [24], and a more efficient model of intruder message creation. We also pointed out the different abstractions we had to make when writing the model, showing that Spin provides less realistic models than other tools. Nevertheless, our analysis showed that the reflection attack is not possible, due to IKEv2's *Initiator* and *Response* flags. Future IKEv2 models should include these flags. Furthermore our analysis confirmed the penultimate authentication flaw and showed that IKEv2-PSK and IKEv2-Child satisfy non-injective agreement in our adversary model.

Although we have analyzed the ability of the specification to meet its security goals, this does not eliminate implementation-level flaws, like buffer overflows and memory leaks. As a consequence, a future work must be performed to detect these flaws on the current and future IKEv2 implementations, e.g. using modern techniques of static analysis and fuzzing.

Acknowledgment

The authors would like to thank Youcef Ech-Chergui for its IKEv2 expertise. In addition, the authors would like to thank the ANSSI [1] and Thomas Given-Wilson for their help with an early version of this paper. The ANSSI provided some technical review, and Thomas Given-Wilson performed some proof-reading.

References

[1] [Online; accessed 04-July-2019]. [Online]. Available: https://wiki.strongswan.org/projects/strongswan/wiki/Fuzzing
[2] Google. [Online]. Available: https://github.com/google/oss-fuzz
[3] C. Meadows, "Analysis of the internet key exchange protocol using the nrl protocol analyzer," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.
[4] C. Cremers, "Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2," in *European Symposium on Research in Computer Security*. Springer, 2011.
[5] A. Project, "Deliverable d6.2: Specification of the problems in the high-level specification language," Tech. Rep., 2003. [Online]. Available: http://www.avispa-project.org/
[6] R. Küsters and T. Truderung, "Using proverif to analyze protocols with diffie-hellman exponentiation," in *22nd IEEE Computer Security Foundations Symposium*, July 2009.
[7] C. Meadows, "The nrl protocol analyzer: An overview," *The Journal of Logic Programming*, 1996.
[8] D. Basin, S. Mödersheim, and L. Viganò, "An on-the-fly model-checker for security protocol analysis," in *Computer Security – ESORICS 2003*, 2003.
[9] C. J. F. Cremers, *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven, Netherlands, 2006.
[10] N. Ben Henda, "Generic and efficient attacker models in spin," in *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*. ACM, 2014.
[11] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001.*, 2001.
[12] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*. Springer Berlin Heidelberg, 2013.
[13] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (IKEv2)," RFC 7296, November 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8019.txt
[14] K. S. and S. K., "Security Architecture for the Internet Protocol," RFC 4301, December 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4301.txt
[15] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, Jun 1992.
[16] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *J. ACM*, Oct. 2017.
[17] R. Gerth, "Concise promela reference," June 1997, [Online; accessed 07-March-2018]. [Online]. Available: http://spinroot.com/spin/Man/Quick.html
[18] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.
[19] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.
[20] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, Mar 1983.
[21] D. Basin, C. Cremers, and C. Meadows, "Model checking security protocols," *Handbook of Model Checking*, 2015. [Online]. Available: http://www-oldurls.inf.ethz.ch/personal/basin/pubs/security-modelchecking.pdf
[22] G. J. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, May 1997.
[23] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *European Symposium on Research in Computer Security*. Springer, 2010.
[24] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings 10th Computer Security Foundations Workshop*, Jun 1997.
[25] C. J. Cremers, S. Mauw, and E. P. de Vink, "Injective synchronisation: an extension of the authentication hierarchy," *Theoretical Computer Science*, 2006.
[26] [Online]. Available: https://gitlab.com/deviation/spin
[27] T. Ninet, A. Legay, R. Maillard, L.-M. Traonouez, and O. Zendra, "The deviation attack: A novel denial-of-service attack against ikev2," in *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-19)*, 2019, to appear.

[1]The ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) is the national cybersecurity agency of France