Model Checking the IKEv2 Protocol Using Spin

Tristan Ninet, Axel Legay, Romaric Maillard, Louis-Marie Traonouez, Olivier Zendra

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 show that the reflection attack is actually not applicable. We further discuss two modifications of the protocol and prove that both of them do overcome the vulnerability the penultimate authentication flaw.

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 [16] 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.

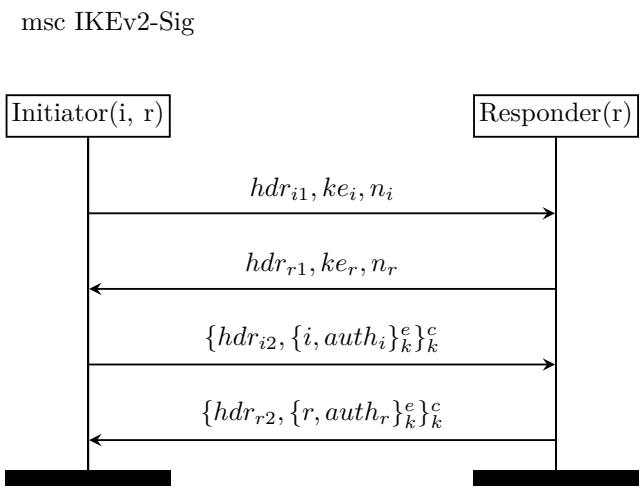
Model checking has been used, to our knowledge, twice to analyze IKEv1 [23, 11] and three times to analyze IKEv2 [26, 19, 11]. Tools that were used are NRL [24], OFMC [7], Scyther [13] and DH-ProVerif [19]. It revealed two authentication vulnerabilities that were found in both IKEv1 and IKEv2: the penultimate 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 the method of Ben Henda [8] for modeling protocols in Promela (Spin's modeling language). We present our IKEv2 model in section IV, and provide its source code so that it can be reused. We explain that Spin is not the most appropriate tool for analyzing security protocols. Other tools such as ProVerif [9] and Tamarin [25] 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 [11], which reported the vulnerability, was missing some payloads that actually prevent the attack. Finally, we discuss in Section VI two possible modifications of the protocol, and formally prove using model checking that each of them eliminates the penultimate authentication flaw.

II. The IKEv2 protocol

The IKEv2 specification [18] 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 [27] 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 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



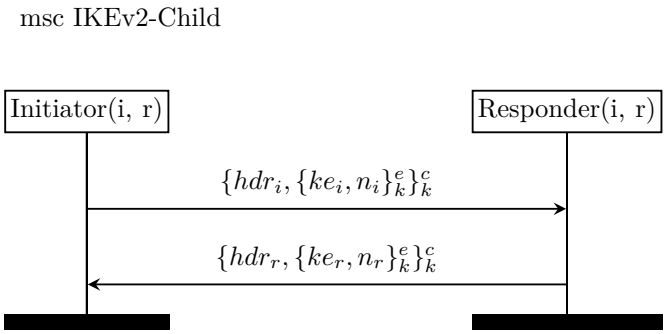
$ke_i = g^{x_i}$ $\quad hdr_{il} = (rf_{il}, if_{il}, mid_{il})$

$ke_r = g^{x_r}$ $\quad hdr_{rl} = (rf_{rl}, if_{rl}, mid_{rl})$

$keymat = k$ $\quad auth_r = \{ke_r, n_r, n_i, k, r\}^s_{prk(r)}$

$k = (ke_i)^{x_r} = (ke_r)^{x_i}$ $\quad auth_i = \{ke_i, n_i, n_r, k, i\}^s_{prk(i)}$

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



$hdr_i = (rf_i, if_i, mid_i)$

$hdr_r = (rf_r, if_r, mid_r)$

$keymat = (ke_i)^{x_r} = (ke_r)^{x_i}$

$ke_i = g^{x_i}$

$ke_r = g^{x_r}$

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

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 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.

To simplify the model checking process, we target 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).

For an agent a, we write $prk(a)$ its private key and $pbk(a)$ its public key. For two agents a and b, we write $psk(a, b)$ their pre-shared key. We write g the Diffie-Hellman generator. For an agent a performing an IKEv2 subprotocol, we use the following notation for payloads that it sends: $(hdr_{al})_{l \in \mathbb{N}_{>0}}$ denotes its IKEv2 headers, ke_a denotes its key-exchange payload, n_a denotes its nonce payload and $auth_a$ denotes its authentication payload. We also write x_a its Diffie-Hellman secret number, which is not sent.

For an agent a performing an IKEv2 subprotocol, we write rf_{al}, if_{al} and mid_{al}, where l is an integer, to respectively denote the Response flag, Initiator flag and message ID fields of the IKEv2 headers that it sends. We simply note them rf_a, if_a and mid_a when the subprotocol contains only one exchange. The Response flag is set to 1 when the message it is in is a response, and the Initiator flag is set to 1 when the message it is in 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.

For a given message msg, we write $\{msg\}^e_k$ the symmetric encryption of msg using key k, $\{msg\}^s_k$ the digital signature of msg using key k (or keyed hash if k is a pre-shared key), and $\{msg\}^c_k$ the message msg in plain text but integrity protected by a checksum using key k. For $q, r \in \mathbb{N}_{>0}$, we write q^r the modular exponentiation (exponentiation in a finite field) of q by r.

Figure 1 shows the message sequence charts (MSC) of the IKEv2-Sig protocol. The IKEv2-PSK MSC can be obtained from it by replacing $prk(i)$ and $prk(r)$ with

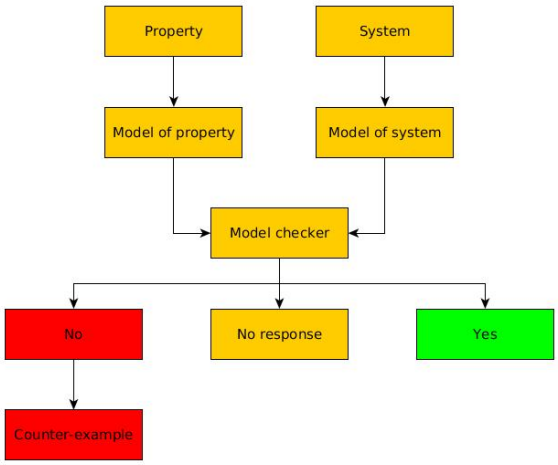


Fig. 3. The model checking workflow. Either proves that a system satisfies a property, or returns a counter-example, or is unable to answer.

$psk(i, r)$. Figure 2 shows the IKEv2-Child MSC. These MSCs contain several simplifications compared to IKEv2's RFC. For example, we do not show all IKEv2 payloads and fields. We show only those that we think have an effect on whether IKEv2 satisfies our properties or not. We explain these simplifications in Section IV.

On these figures, $Initiator(i, r)$ means that agent i is taking the role of initiator and wants to perform the protocol with agent r. $Responder(r)$ means that agent r is taking the role of responder and can perform the protocol with whatever agent correctly authenticates itself and is trusted by r. Finally, $Responder(i, r)$ means that agent r is taking the role of responder and can only perform the protocol with agent i.

Furthermore, in the IKEv2-Child MSC, the term k denotes a key that is shared by the initiator and the responder before running the subprotocol. It represents the set of keys SK_ai, SK_ar, SK_ei, SK_er defined in the IKEv2 RFC, which are used to protect the IKE SA in which the subprotocol is performed. Finally, the term $keymat$ represents the term $KEYMAT$ defined in the IKEv2 RFC, which is a cryptographic value from which are derived all keys protecting the traffic SA negotiated through the subprotocol. It is the keying material of which IKEv2 aims to guarantee the secrecy.

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. We focus on Linear Temporal Logic (LTL) [20] properties. LTL is built upon boolean logic with the addition of time indicators like always (denoted $\Box$) and eventually (denoted $\Diamond$). The linear-time property is verified on each and every execution trace of the model. The model is written in a specific modeling language, such as Applied Pi Calculus [3] or Promela [15]. 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. Figure 3 sums up the steps of the model checking process. The principles of model checking are explained in great details in [4, 10].

We want a security protocol to be secure even in the presence of an adversary, that can intercept, drop, replay, learn from and build messages. To formalize this, Dolev and Yao first defined [14] 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, like secrecy and authentication. Different techniques can be used to represent and explore the model, which model checkers abstract away: Models can be explored in a forward or backward manner. One can allow a finite or infinite number of runs (i.e. protocol executions). States can be represented explicitly or symbolically. Finally, abstractions can be used to trade completeness for efficiency. A thorough state-of-the-art of model checking security protocols is depicted in [6].

B. Related work

In 1999, Meadows finds two authentication weaknesses in IKEv1 [23], 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 [26]. 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: the key confirmation.

In 2009, Kusters and Truderung use their tool DH-ProVerif to verify IKEv2 [19]. Their analysis seems to confirm the penultimate authentication flaw.

In 2010, Cremers performs an extensive analysis of IKEv2 [11] 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, [11] finds that the reflection attack that was noted for IKEv1 is also possible on IKEv2.

C. The Spin model checker

Spin [17] is a general-purpose explicit-state model checker. It takes Promela [15] 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-sate), 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 by Ben Henda in [8]. We describe this method later in Section IV.

We observe that, even with the Ben Henda method and our improvements to this method, Spin remains not the most appropriate tool for model checking security protocols: Our model only allows up to two sessions in an execution trace, and a maximum number of tree agents running these sessions. Furthermore, intruder knowledge is bounded to one message and the complexity of messages that the intruder can send over the network is limited. Other tools that are specialized in security protocol model checking do not suffer from these limitations. Such tools include ProVerif [9], DH-ProVerif [19], Scyther [13] and Tamarin [25].

IV. Modeling IKEv2 in Promela

As said in Section III, verifying IKEv2 using Spin involves modeling three different concepts in Promela: the protocol itself, the adversary model and the properties. The Promela code we wrote is available at [2].

A. Modeling the protocol in Promela

Ideally, we would like to write a Promela code that represents the exact behaviour of IKEv2 as defined in its RFC. However, as pointed out in Section II, IKEv2 is far too complex to be fully modeled. For this reason, we choose to model only a subset of IKEv2 that we think satisfies the same security properties as the full-blown protocol. We say that we model a reduced form of IKEv2. For example, we do not include the traffic selector (TS) payload in our model, because we think that it has no effect on whether IKEv2 satisfies our properties or not. Indeed, the TS payloads are not used in any cryptographic operation. They neither play a role in key generation nor in authentication. This abstraction is error prone and is the reason why [11] finds an attack that does not exist (the reflection attack). We discuss this in Section V.

The MSCs of Section II represent the reduced protocols that we want to model in Promela. We make the following simplifications compared to the RFC of IKEv2. As said above, we do not include all payloads and fields, because only some of them are relevant to the properties we want to verify. In addition, in the IKEv2 RFC, we have $k = prf(n_i|n_r, g^{x_i x_r})$, where prf a is pseudo-random function and | denotes concatenation. In our model, we simply have $k = g^{x_i x_r}$. Finally, we abstract away key derivation: In the RFC, the SK_d, ..., SK_pr keys are derived from $SKEYSEED$, and the $KEYMAT$ value is derived from SK_d. In our model, those values are all represented by a single term k.

An other 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 of Ben Henda, explained in [8]. In this method, roles are implemented by processes, and the network by a synchronous channel. We define Promela mtype constants (a Promela type of variable) describing agents, keys and a small set of values that agents can use as nonces during the protocol. As an example, in the code below, taken from the initiator process in our model, the initiator sends an IKE_AUTH request. Just before, we execute the $Inirunning(i, r)$ macro, which sets variable *inirunningab* to 1 if $i = A$ and $r = B$, variable *arunning* to 1 if $i = a$ and variable *brunning* to 1 if $i = b$. These variables are used in our properties expression, which we present later in this Section.

```
Inirunning(i, r, k);
Comm!M3, k, FR0, FI1, MID1, k, i, authkeyi,
    k, i, kei, ni, nr;
```

Listing 1. The initiator sending an IKE_AUTH request

Comm is the name of the channel over which the message is sent. The $M3$ term indicates the type of message (here IKE_AUTH request). The $FR0$ and $FI1$ terms denote Responder and Initiator flags respectively set to 0 and 1, and the $MID1$ term represents a message ID set to 1. The other terms can easily be understood since they resemble the terms we define in Section II. On reception of the message, everything after k key is interpreted as encrypted by k, and everything after *authkeyi* key is interpreted as a signature computed with our Promela equivalent of $prk(i)$, or a keyed hash computed with Promela equivalent of $psk(i, r)$.

We improve the Ben Henda method in order to fit the needs of IKEv2. Ben Henda does not provide a way to model a Diffie-Hellman exchange, so we create it. It requires adding a deduction step when adding a message to the intruder's knowledge. Indeed, when the intruder learns a payload, we now need to check whether it can deduce a key by modular exponentiation.

Even with the Ben Henda method, the nature of Promela inherently adds a layer of abstraction to the modeling process. In particular, 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). Nevertheless, such a model can capture a large class of attacks. In the code below, taken from our model's *init* process, 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.

```
if
:: run Initiator(A, B)
:: run Initiator(A, C)
fi;

run Responder(B);
```

Listing 2. Instantiation of two runs

B. Modeling the adversary model in Promela

The property we verify strongly depends on the capabilities given to the intruder, during verification. Consequently, Basin and Cremers decided in [5] to split each security property into an adversary model and an atomic property. We follow the same principle in our model.

Basin and Cremers translate several adversary models from the literature into their own formalism. We implement two of them in our model: the external Dolev-Yao model (AdvEXT) and the internal Dolev-Yao model (AdvINT). AdvEXT is a minimalistic symbolic model. As explained in Section III, 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.

In AdvINT, the intruder has the same capabilities as in AdvEXT, plus the LKRothers capability. The latter allows it 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. For example, in the Aliveness property defined later in this Section, agents A and B are mentioned. In AdvINT, the intruder thus learns the private key of agent C. AdvINT corresponds to the model used by Lowe to find his famous attack on the Needham-Schroeder protocol [22]. A more precise definition of LKRothers is given in [5].

In the Ben Henda method, the intruder is modeled by a process (just like the initiator and responder roles), that can non-deterministically receive from the channel, forge a new message and send it, or replay a saved message. The intruder's knowledge is modeled by a boolean vector (called *Knows*) indexed by all the protocol constants. A memory of exactly one message is given to the intruder. This is an abstraction of the adversary model. Unfortunately, increasing this memory quickly leads to a path explosion.

We improve the Ben Henda method concerning the adversary model. To allow the intruder forging messages, Ben Henda uses a general-purpose $RandMessage$ macro that chooses one value among all mtype constants, followed by an $IsValidMessage$ macro that checks the forged message's validity in regard to the intruder knowledge. He proposes more efficient definitions of $RandMessage$, but none of them are efficient enough for our analysis. Our $Randm1m2message$ macro (our equivalent of $RandMessage$ but only for IKE_SA_INIT) directly chooses for each payload a value among mtype constants whose type fits the payload. This greatly limits path explosion, without losing any realistic behaviour. We provide the code of $Randm1m2message$ below.

```
inline Randm1m2message(p1, p2, p3, p4, p5,
    p6, p7, p8, p9, p10, p11, p12, p13)
{
    /* The message is M1 or M2. We set the
    Response and Initiator flags. */
        if
        :: p1 = M1; p2 = FR0; p3 = FI1
        :: p1 = M2; p2 = FR1; p3 = FI0
        fi;

        /* The message ID is 0. */
        p4 = MID0;

        /* We set the Key Exchange payload. */
        if
        :: Knows[KEA] -> p5 = KEA
        :: Knows[KEB] -> p5 = KEB
        :: Knows[KEC] -> p5 = KEC
        fi;

        /* We set the Nonce payload. */
        if
        :: Knows[NA] -> p6 = NA
        :: Knows[NB] -> p6 = NB
        :: Knows[NC] -> p6 = NC
        fi;

    /* All other payloads are empty: they are
    only used for the M3 and M4 messages. */

        p7 = NULL;
        p8 = NULL;
        p9 = NULL;
        p10 = NULL;
        p11 = NULL;
        p12 = NULL;
        p13 = NULL;
}
```

Listing 3. The Randm1m2message inline function makes the intruder forge an IKE_SA_INIT message

Note that the range of messages that the intruder can send over the network is limited. In Tamarin and ProVerif adversary models, any payload in any message

sent by the intruder may contain any constants known by the attacker, encrypted any number of times, using several encryption algorithms and any key known by the attacker. In our model, we restricted for each payload the range of constants that may be sent by the intruder to a pre-selected set of constants. For example, m1 and m2 messages' fifth payload may only contain constants KEA, KEB or KEC.

C. Modeling the properties in Promela

Ideally, we would like to verify the exact properties the protocol claims to guarantee. However, security properties can be quite vague. In particular, the "mutual authentication" mentioned in [18] is an unclear notion. For this reason, researchers have split it into several definitions [21, 12]. 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.

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 agreement are authentication properties and were first defined by Lowe in [21]. 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. 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. "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.

Note that agreement implies weak agreement, which in turn implies aliveness. There are stronger authentication properties that we could verify. The injective agreement property, for example, adds to agreement the condition that there is only one matching session. Our model allowing only one session per role in a trace, it would not be relevant to check injective properties.

For IKEv2-Sig and IKEv2-PSK, we make parties agree upon the term *keymat*. For IKEv2-Child, however, we cannot guarantee to the responder that the initiator has computed *keymat* since the initiator has already sent its last message when it computes *keymat*. Therefore, for agreement in IKEv2-Child, we only guarantee to the responder that the initiator has computed *kei*. To the initiator we guarantee that the responder has computed *keymat*

We make some improvements to the Ben Henda method in the definition of our properties as well. Ben Henda verifies only one variant of authentication that can be seen as weak agreement where the peer has the correct correct role. We implement aliveness, weak agreement and agreement. In addition, our model of secrecy is more faithful to the intuitive definition of secrecy: it is not enough to check that the mtype constant we create for *keymat* is not known to the intruder (as [8] does), one needs to check that any constant that an agent having completed the protocol considers as its *keymat* value, is not known to the intruder.

The code below, taken from our model, defines the properties' invariants. If one of them becomes false during execution, then the corresponding property is violated. The values of the variables appearing in the invariants are set during protocol execution. Their goal is to keep track, for each agent, where it is at in its protocol execution, as well as to whom it believes it is talking and what value it has computed for *keymat*. We define the terms $\{ini, res\}\{running, commit\}ab$ as in the work of Ben Henda [8]: *inicommitab* (resp. *rescommitab*) means that A (resp. B) has completed the protocol as an initiator (resp. responder), apparently with B (resp. A). In the same way, *inirunningab* (resp. *resrunningab*) means that A (resp. B) has been running the protocol (in the sense defined earlier in this Section) as an initiator (resp. responder), apparently with B (resp. A). The term *arunning* (resp. *brunning*) means that A (resp. B) has been running the protocol. The value of the term *ka* (resp. *kb*) is the mtype constant that A (resp. B) considers as its *keymat* key. Finally, as explained earlier in this Section, if $Knows[ka]$ is true, then the intruder knows the term *ka*.

```
# define Invsecrecy ( \
    (!inicommitab || !rescommitba || \
        sta != NULL && !Knows[sta]) && \
    (!rescommitab || !inicommitba || \
        stb != NULL && !Knows[stb]) )

# define Invaliveness ( \
    (!inicommitab || brunning) && \
    (!rescommitab || arunning) && \
    (!inicommitba || arunning) && \
    (!rescommitba || brunning) )

# define Invweakagree ( \
    (!inicommitab || resrunningab || \
        inirunningba) && \
    (!rescommitab || inirunningab || \
        resrunningba) && \
    (!inicommitba || resrunningba || \
```

```
        inirunningab) && \
    (!rescommitba || inirunningba || \
        resrunningab) )

# define Invagree ( \
    (!inicommitab || resrunningab && \
        atca != NULL && atca == atrb) && \
    (!rescommitab || inirunningab && \
        atcb != NULL && atcb == atra) && \
    (!inicommitba || resrunningba && \
        atcb != NULL && atcb == atra) && \
    (!rescommitba || inirunningba && \
        atca != NULL && atca == atrb) )
```

Listing 4. Definition of our properties's invariants in Promela

V. Analysis results

Our Promela code and the exact Spin commands we used are available at [2]. 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 one notable result: it refutes the reflection attack that was found by previous analyses.

In [11], Cremers 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, [11] 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 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.

Cremers 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" [11]. 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. Nevertheless, an important protocol such as IKEv2 should satisfy strong forms of authentication.

VI. Counter-measures

A way to fix the protocol is to make the IDr payload mandatory in IKE_AUTH request. The message sequence chart on figure 5 shows the modified protocol. The drawback of this modification is that now the initiator sends IDr to the responder before the responder has authenticated itself. Therefore if the responder is an attacker (we then speak of active attacker), the initiator identity is revealed.

To fix IKEv2-Sig without disclosing the responder ID to an active attacker, we add a third exchange, called key confirmation. The key confirmation exchange is made of an empty INFORMATIONAL request and an empty INFORMATIONAL response. INFORMATIONAL messages are a type of message in IKEv2 that are e.g. used as keep-alive messages. The message sequence chart on figure 6 shows the modified protocol. Key confirmation has already been proposed by Basin et al. in [26]. However this modification has the drawback of adding a whole exchange to the protocol and therefore increasing its cost in time, memory, and computation power.

To verify that these modifications remove the penultimate authentication flaw, we apply them to our Promela models. We define four new subprotocols:

IKEv2-PSK-IDr consists of one IKE_SA_INIT exchange and one IKE_AUTH exchange. It uses PSK authentication and the IDr counter-measure.

IKEv2-Sig-IDr consists of one IKE_SA_INIT exchange and one IKE_AUTH exchange. It uses Signature authentication and the IDr counter-measure.

IKEv2-PSK-conf consists of one IKE_SA_INIT exchange, one IKE_AUTH exchange and one KEY_CONF exchange. It uses PSK authentication and key confirmation.

IKEv2-Sig-Conf consists of one IKE_SA_INIT exchange, one IKE_AUTH exchange and one KEY_CONF exchange. It uses Signature authentication and key confirmation.

We apply the counter-measures to IKEv2-PSK as well, to verify that they do not make it lose any guarantee. With these modifications, we find that these four subprotocols satisfy all our properties in all our adversary models: IKEv2's current security properties are preserved and the protocol gains stronger authentication guarantees. The

Property	Adversary model	IKEv2-Sig	IKEv2-PSK	IKEv2-Child
Secrecy	AdvEXT	✓	✓	✓
	AdvINT	✓	✓	✓
Aliveness	AdvEXT	✓	✓	✓
	AdvINT	✓	✓	✓
Weak agreement	AdvEXT	✗	✓	✓
	AdvINT	✗	✓	✓
Agreement	AdvEXT	✗	✓	✓
	AdvINT	✗	✓	✓

Fig. 4. Analysis of IKEv2 using Spin. We write ✓when a subprotocol satisfies a property in a specific 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.

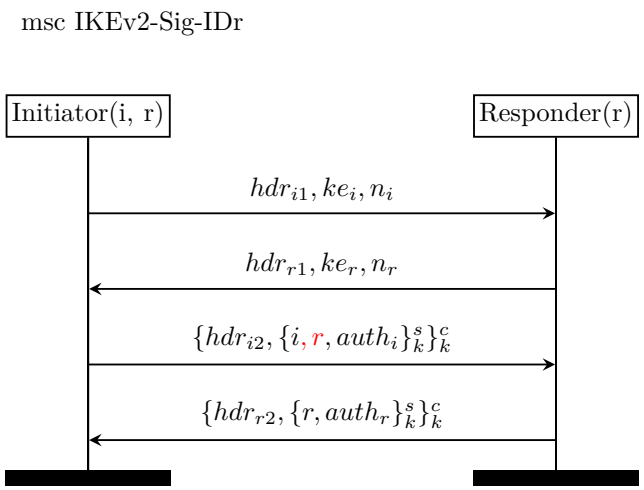


Fig. 5. The IKEv2-Sig-IDr protocol. We make IDr (r in the figure above) payload mandatory in the IKE_AUTH request and modify its processing by Responder.

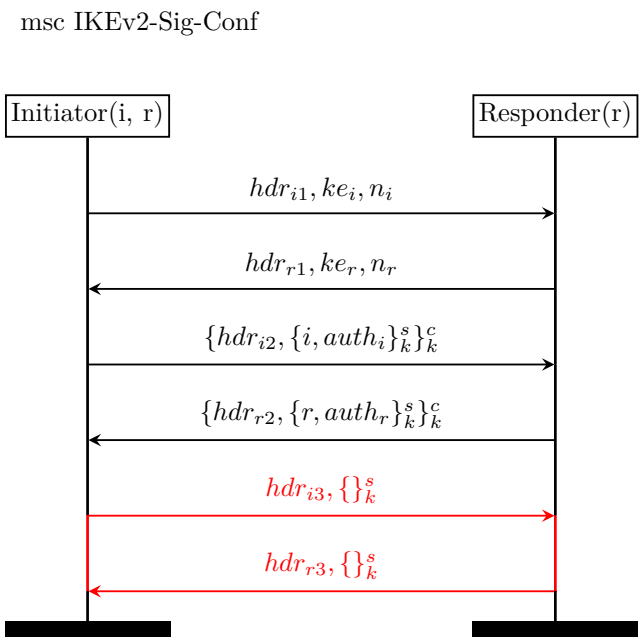


Fig. 6. The IKEv2-Sig-conf protocol. We add the key confirmation exchange. Connections are installed only after this exchange.

modified Promela models can be found with our other models at [2].

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 Ben Henda [8] with a model of Diffie-Hellman exponentiation, an implementation of aliveness, a better implementation of secrecy and a more efficient model of intruder message creation. We also pointed out the different abstractions we made when writing the model, showing that Spin is not state-of-the-art for analyzing security protocols. 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. Futhermore our analysis confirmed the penultimate authentication flaw. As a counter-measure we discussed two possible modifications of the protocol and proved that both of them do overcome the vulnerability.

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 Thomas Given-Wilson for its help with proof-reading an early version of this

paper, and Youcef Ech-Chergui for its IKEv2 expertise. In addition, the authors would like to thank the ANSSI [1] for their technical review of an early version of this paper.

References

[1] [Online; accessed 11-February-2018]. 2017. url: https://en.wikipedia.org/wiki/ARP_spoofing.

[2] url: https://gitlab.com/deviation/spin.

[3] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: J. ACM (Oct. 2017).

[4] Christel Baier and Joost-Pieter Katoen. Principles of model checking. MIT press, 2008.

[5] David Basin and Cas Cremers. "Modeling and analyzing security in the presence of compromising adversaries". In: European Symposium on Research in Computer Security. Springer. 2010.

[6] David Basin, Cas Cremers, and Catherine Meadows. "Model checking security protocols". In: Handbook of Model Checking (2015). url: http://www-oldurls.inf.ethz.ch/personal/basin/pubs/security-modelchecking.pdf.

[7] David Basin, Sebastian Mödersheim, and Luca Viganò. "An On-the-Fly Model-Checker for Security Protocol Analysis". In: Computer Security – ESORICS 2003. 2003.

[8] Noomene Ben Henda. "Generic and efficient attacker models in spin". In: Proceedings of the 2014 International SPIN Symposium on Model Checking of Software. ACM. 2014.

[9] B. Blanchet. "An efficient cryptographic protocol verifier based on prolog rules". In: Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001. 2001.

[10] Edmund M Clarke, Orna Grumberg, and Doron Peled. Model checking. MIT press, 1999.

[11] Cas Cremers. "Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2". In: European Symposium on Research in Computer Security. Springer. 2011.

[12] Cas JF Cremers, Sjouke Mauw, and Erik P de Vink. "Injective synchronisation: an extension of the authentication hierarchy". In: Theoretical Computer Science (2006).

[13] Casimier Joseph Franciscus Cremers. Scyther: Semantics and verification of security protocols. Eindhoven University of Technology Eindhoven, Netherlands, 2006.

[14] D. Dolev and A. Yao. "On the security of public key protocols". In: IEEE Transactions on Information Theory (Mar. 1983).

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

[15] Rob Gerth. Concise Promela Reference. [Online; accessed 07-March-2018]. June 1997. url: http://spinroot.com/spin/Man/Quick.html.

[16] Google. url: https://github.com/google/oss-fuzz.

[17] G. J. Holzmann. "The model checker SPIN". In: IEEE Transactions on Software Engineering (May 1997).

[18] Charlie Kaufman et al. Internet key exchange protocol version 2 (IKEv2). RFC 7296. Nov. 2014. url: http://www.rfc-editor.org/rfc/rfc8019.txt.

[19] R. Küsters and T. Truderung. "Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation". In: 22nd IEEE Computer Security Foundations Symposium. July 2009.

[20] Linear Temporal Logic. Sept. 2017. url: http://spinroot.com/spin/Man/ltl.html.

[21] Gavin Lowe. "A hierarchy of authentication specifications". In: Proceedings 10th Computer Security Foundations Workshop. June 1997.

[22] Gavin Lowe. "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR". In: Tools and Algorithms for the Construction and Analysis of Systems. 1996.

[23] 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.

[24] Catherine Meadows. "The NRL Protocol Analyzer: An Overview". In: The Journal of Logic Programming (1996).

[25] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: Computer Aided Verification. Springer Berlin Heidelberg, 2013.

[26] AVISPA Project. Deliverable D6.2: Specification of the Problems in the High-Level Specification Language. Tech. rep. 2003. url: http://www.avispa-project.org/.

[27] Kent S. and Seo K. Security Architecture for the Internet Protocol. RFC 4301. Dec. 2005. url: https://www.rfc-editor.org/rfc/rfc4301.txt.