The Deviation Attack: A Novel Denial-of-Service Attack Against IKEv2

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

The Deviation Attack: A Novel Denial-of-Service Attack Against IKEv2

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

***Abstract*—In previous analyses, IKEv2 has been shown to possess two authentication vulnerabilities that were considered not exploitable. In this paper, we analyze the protocol specification using the Spin model checker, and prove that in fact the first vulnerability does not exist. In addition, we show that the second vulnerability is exploitable by designing and implementing a novel slow Denial-of-Service attack, which we name the Deviation Attack. We explain the attack's requirements, discuss possible counter-measures and propose a modification of the protocol that we prove eliminates the vulnerability.**

I. Introduction

Virtual Private Networks (VPN) have been used for decades by companies, governments and people. VPNs allow connecting two or more distant IP entities as if they were in a single Local Area Network (LAN). By "entity" we either mean a network or a single machine. Since the network between the entities may be non-trusted, connecting the entities often implies performing some encryption and some address translation on the IP traffic between them. In a company, an entity can among other things denote a traveling employee or a working site. In the military domain, an entity can among other things denote a soldier on the battle field or a military base. More recently, we have observed the rise of commercial VPN services for the greater public. Companies offer individual consumers to enter one of the companies' VPNs so that the consumers can browse the Internet as if they were in one of the companies' LANs. Many people use VPN services to spoof their location in order to access services denied to them based on their geographical location. Many people also use them to evade censorship, since VPN services allow them to hide the content of their packets between them and the VPN service company. VPNs thus have become a core technology in the modern secure Internet, and it is crucial that VPNs remain secure.

A VPN can be achieved using multiple technologies, among which the Internet Protocol security (IPsec) architecture [1] is widely used. Moreover, Internet Key Exchange version 2 (IKEv2) [2] is one of the main protocols used to set up IPsec VPNs. IKEv2 aims at guaranteeing mutual authentication of two peers, and at automatically generating the shared secret that will be of the communication's security warrant. Thus a secure VPN relies upon the security of IKEv2, and it is of the utmost importance that IKEv2 be secure.

A security protocol such as IKEv2 can possess two types of vulnerabilities: implementation vulnerabilities and specification vulnerabilities.

Implementation vulnerabilities encompass all flaws in the implementation. For example, vulnerabilities allowing buffer overflows are implementation vulnerabilities. An appropriate approach to avoid implementation vulnerabilities is to use modern automated testing techniques like fuzzing. In fact, the developers of the strongSwan IKEv2 implementation have recently announced [3] that part of their code base is now fuzzed using Google's OSS-Fuzz [4] 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.

Specification vulnerabilities take in all weaknesses in the protocol specification itself. These weaknesses lead to different attacks, such as Man in the Middle (MitM) attacks and reflection attacks. An efficient way to find specification vulnerabilities is to use automated techniques, such as model checking. Model checking allows to detect specification flaws early in the development process, which is much cheaper than when specification flaws are discovered during implementation, validation, or after. 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 only rely 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. IKEv2 is designed so as to work even in the presence of Network Address Translation (NAT) between the peers. IKEv2 possesses not less than twenty-nine types of notification and error messages. Thus it is easier to use an automated process to verify IKEv2.

Model checking has been used, to our knowledge, twice to analyze IKEv1 [5], [6] and twice to analyze IKEv2 [6], [7]. It revealed two authentication vulnerabilities that were found in both IKEv1 and IKEv2: the penultimate authentication flaw and the reflection attack. However, they were not considered a serious concern because they did not question the secrecy of the shared key generated by IKEv2.

In this paper, we start by giving some background on IKEv2 in Section II. We then use the Spin model checker [8] to analyze the IKEv2 specification. Our analysis of IKEv2 is described in Section III. To analyze IKEv2 using Spin, we improve the method of Ben Henda [9] for modeling protocols in Promela (Spin's modeling language), and provide the source code of our model so that our model can be reused by others. As a result, we show that the reflection attack has no practical existence against IKEv2. The model of Cremers, who

reported the vulnerability in [6], was missing some payloads that actually prevent the attack.

In Section IV, we design a novel Denial-of-Service (DoS) attack against IKEv2 that exploits the penultimate authentication flaw. We call the novel DoS attack the Deviation Attack. The Deviation Attack bypasses all measures that were introduced in IKEv2 to resist DoS attacks. We thoroughly discuss the Deviation Attack's flow and details, and calculate the precise quantities that trigger the attack. To demonstrate the Deviation Attack very concretely, we implement it in Section V; thereby attacking an open-source implementation of IKEv2. We experimentally verify our expression of the triggering quantities through this experiment, and provide the source code so that the reader can easily reproduce the attack.

Finally, in Section VI, we explore a number of ways to protect the implementations only using what the current protocol specification has to offer. However, we find only mitigations or incomplete workarounds. We therefore tackle the problem at a higher level: We propose two possible inexpensive modifications of the protocol, and formally prove using model checking that each of them eliminates the attack.

For ethical reasons we informed our country's national security agency about the existence of the Deviation Attack. The security agency gave us some technical feedback as well as its approval for publishing the attack.

II. IKEv2

A. Overview

Internet Key Exchange version 2 (IKEv2) is the authenticated key-exchange protocol used in the Internet Protocol security architecture (IPsec). Its specification is managed by the Internet Engineering Task Force (IETF), and the current RFC is RFC 7296 [2]. The goal of IKEv2 is to allow two peers to dynamically negotiate cryptographic algorithms and keys in order to set up an IPsec communication. As such, it aims to guarantee mutual authentication of the peers and secrecy of the negotiated keys. An open-source implementation of IKEv2 is strongSwan.

The IPsec architecture provides security at the networking layer. It is defined in RFC 4301 [1]. IPsec defines a framework to establish Virtual Private Networks (VPN). Depending on the underlying security protocol that is used, IPsec provides either integrity/authentication protection (AH protocol) or confidentiality and integrity/authentication protection (ESP protocol) of IP packets that are exchanged between two peers. IPsec also natively brings some protection against replay attacks (using this protection is however at the discretion of the receiver of an IPsec protected packet). To protect packets, IPsec peers set up Security Associations (SA) between them. A Security Association is a set of security parameters and keys on which two peers agree.

Setting up an SA requires that the peers share some encryption keys and involves adding entries to their Security Association Database (SAD). The keys can be manually put into the peers' databases, but the maintenance of such a configuration becomes cumbersome when the number of peers grows. To ease management of large VPN setups, it is much more efficient to rely upon dynamic negotiation of cryptographic material as defined by the IKEv2 protocol.

We call an *exchange* the association of a request message and a response message. IKEv2 consists of three main exchanges. The IKE_SA_INIT exchange performs initial setup of an IKE SA that will be used to protect subsequent exchanges of the IKEv2 protocol. During this exchange, peers agree upon cryptographic algorithms that should be used to protect further IKEv2 exchanges and establish some common cryptographic material by running a Diffie-Hellman protocol. The IKE_AUTH exchange authenticates the peers, validates the IKE SA and sets up a traffic SA (or Child SA). The authentication can be done using pre-shared key (PSK) or digital signature. It is a counter-measure to the Man-in-the-Middle attack that can be performed against the Diffie-Hellman exchange of IKE_SA_INIT. Finally, the CREATE_CHILD_SA exchange has two different purposes. First, it can be used for rekeying an IKE SA, i.e. for replacing an old IKE SA with a new one. In this case, its payloads for performing a Diffie-Hellman exchange (the key-exchange payloads) are mandatory. Second, it can be used to create a new traffic SA, or to rekey an existing one. In this case, the key-exchange payloads are optional. If not provided, the new keys are simply derived from the IKE SA's keys. Using the key-exchange payloads provides Perfect Forward Secrecy for the new traffic SA's keys, i.e. their secrecy will not be impacted by the compromise of the IKE SA's keys. The CREATE_CHILD_SA exchange is performed multiple times during the lifetime of an IKE SA.

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 known only 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.

B. Payloads

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:

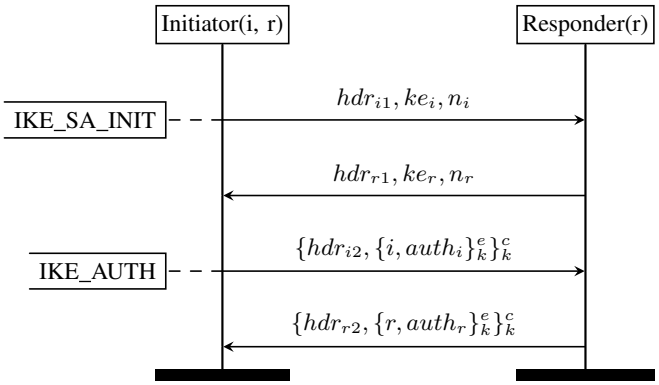
IKEv2-Sig consists of one IKE_SA_INIT exchange and one IKE_AUTH exchange. It uses digital signature authentication.

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.

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.

msc IKEv2-Sig



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

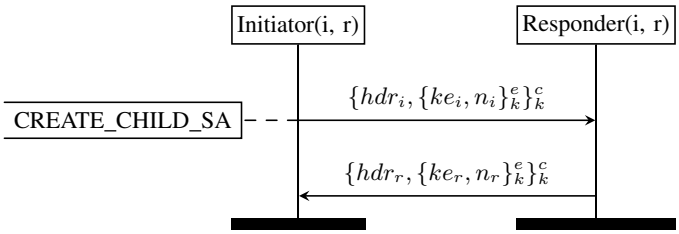
$ke_r = g^{x_r}$ $\quad hdr_{rl} = (rf_{rl}, if_{rl}, mid_{rl})$

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

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

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

msc IKEv2-Child



$hdr_i = (rf_i, if_i, mid_i)$ $\quad ke_r = g^{x_r}$

$hdr_r = (rf_r, if_r, mid_r)$ $\quad ke_i = g^{x_i}$

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

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

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.

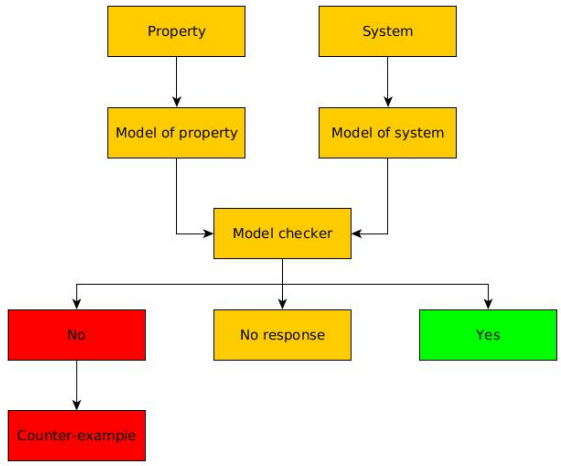


Fig. 3. The model checking workflow. Either proves that a system satisfies a property, or returns a counter-example, or is unable to answer.

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 $psk(i, r)$. 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 III-C.

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. Model checking IKEv2

A. 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) [10] 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 [11] or Promela [12]. 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 [13], [14].

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 [15] 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 [16].

b) Related work: In 1999, Meadows finds two authentication weaknesses in IKEv1 [5], using the NRL protocol analyzer. The first one is a reflection attack, and the second one is called the penultimate authentication flaw. We will explain these later.

In 2003, IKEv2 is formally verified in the context of the AVISPA project [7]. The authors find that IKEv2 also possesses the penultimate authentication flaw. However, they say that it cannot be exploited for further purposes. They propose a counter-measure anyway, that we will present in Section VI: the key confirmation.

Tool	Bounded/unbounded	Uses abstractions
ProVerif	Both	Yes
Scyther	Both	No
Spin	Bounded	No

Fig. 4. Some differences between ProVerif, Scyther and Spin, that mattered in our choice.

In 2010, Cremers performs an extensive analysis of IKEv2 [6] using the Scyther tool. He confirms that IKEv2 possesses the penultimate authentication flaw and finds that the reflection attack that was noted for IKEv1 is also possible on IKEv2.

B. Tool selection

We considered using several tools for model checking IKEv2. Candidates were ProVerif [17], Scyther [18] and Spin [8].

ProVerif is a rewriting system of horn clauses. Its modeling language is a typed variant of pi calculus that includes control flow. The intruder is implicit and corresponds to an external Dolev-Yao (AdvEXT) (see Section III-C for the definition of AdvEXT). The rewriting paradigm makes it very easy to implement cryptographic primitives and the Diffie-Hellman protocol. However, in order to terminate, ProVerif uses abstractions and is therefore not complete: there are situations in which it can neither prove nor disprove a property.

Scyther works with trace patterns. It uses backward search to determine if a property's trace pattern is realizable or not. If not, the property is satisfied, and vice versa. It can work either in a bounded or in an unbounded fashion. Scyther does not use any abstraction.

Spin is a general-purpose explicit-state model checker. It takes Promela [12] 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 [9]. We describe this method later in Section III-C. Spin is a bounded model checker and can therefore verify properties only for a finite number of sessions. It does not use any abstraction.

Table 4 sums up the main differences between the tools. We first tried ProVerif, that seemed to be the most appropriate tool for model checking of security protocols as it is well fit for cryptographic primitives description. Unfortunately, our tests quickly showed that the abstraction techniques used by the tool made it impossible to come to any conclusion regarding the properties of the IKE-Child subprotocol. We were left with two candidates, one of which (Scyther) had already been used for IKEv2 analysis [6]. It appeared sound to rely on a different tool for our work, in order to get an alternative

assessment of IKEv2 security properties. In addition, since IKEv2 is a complex protocol, it was likely that we would have to bound the number of sessions for the analysis to terminate. We thus did not need the ability of Scyther to represent an unbounded number of sessions. Finally, Spin is 25 years old, well-maintained and robust.

C. Writing our Promela model

As said in Section III-A, 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 [19].

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-B, 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 reduction is error prone and is the reason why Cremers finds an attack in [6] that does not exist (the reflection attack). We discuss this in Section III-D.

The MSCs of Section II-B represent the reduced protocols we want to model in Promela. We make the following simplifications made 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 [9]. 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.

Listing 1. The initiator sending an IKE_AUTH request

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

$Comm$ 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-B. 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 make an improvement to 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 reduction to the modeling process. In particular, because Spin is bounded and because of the constraints in time and memory, we only allow up to two runs in an execution trace and only 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 runs: 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.

Listing 2. Instantiation of two runs

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

run Responder(B);
```

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 [20] to separate 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-A, 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 than 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 [21]. A more precise definition of LKRothers is given in [20].

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 a reduction of the adversary model. Unfortunately, increasing this memory quickly leads to a path explosion.

We make an improvement to the Ben Henda method concerning the adversary model. To make the intruder forge a message, 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 $Randm1m2message$ in Appendix A.

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 [2] is an unclear notion. For this reason, researchers have split it into several definitions [22], [23]. 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.

Aliveness and weak agreement are authentication properties and were first defined by Lowe in [22]. 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 weak agreement implies aliveness. There are stronger authentication properties that we could verify. The *agreement* property, for example, adds to weak agreement the condition that A and B agree on the value of some terms (e.g. some exchanged payloads). However, previous analyses [6] have shown that IKEv2-Sig does not satisfy weak agreement, so there is no point in verifying stronger properties for IKEv2-Sig. As for IKEv2-PSK and IKEv2-Child, we choose to focus on aliveness and weak agreement in this paper.

We make some improvements to the Ben Henda method in the definition of our properties as well. Ben Henda does not provide any model for the aliveness property, which we do. 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, 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 [9]: $inicommitab$ (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.

Listing 3. Definition of our properties's invariants in Promela

```
# define Invsecrecy ( (!inicommitab || ka != NULL \
      && !Knows[ka]) && (!rescommitab \
      || kb != NULL && !Knows[kb]) )
# define Invaliveness ( (!inicommitab || brunning) \
      && (!rescommitab || arunning) )
# define Invweakagree ( (!inicommitab \
      || resrunningab) && (!rescommitab \
      || inirunningab) )
```

D. Results of our IKEv2 analysis

Our Promela code and the exact Spin commands we used are available at [19]. We present our analysis results in table 5. Allowing only two runs 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

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

Fig. 5. 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.

notable result: it refutes the reflection attack that was found by previous analyses.

In [6], 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 set it up with itself, the other agent not even being alive.

However, Cremers omitted 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 and weak 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.

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. We explain the penultimate authentication flaw in Appendix B. This vulnerability is not a full violation of the intuitive definition of authentication, because there is no actual impersonation and secrecy is still satisfied. Still, it allows a Denial-of-Service attack that we present in the next Section.

IV. The Deviation Attack

A. Preliminaries

We assume the existence of $N + M + 1$ IKEv2 parties called *(Initiator$_i$)$_{1\leq i\leq N}$*, *(Responder$_i$)$_{1\leq i\leq M}$*, *Probe*, and *Victim*. Each party may be either an IKEv2 endpoint (also called host) or a gateway. Figure 6 presents the attack scenario. On this figure and throughout the paper, we use the term *m1* as an abbreviation for IKE_SA_INIT request. We define in the same way the terms *m2*, *m3*, and *m4*.

(Initiator$_i$)$_{1\leq i\leq N}$, (Responder$_i$)$_{1\leq i\leq M}$, and Victim are connected by a network *Net*$_1$. Probe and Victim are connected by a network *Net*$_2$. Net$_1$ and Net$_2$ may be the same network and may be the Internet. We write $(I_i)_{1\leq i\leq N}$, $(R_i)_{1\leq i\leq M}$ and V the Net$_1$ IP addresses.

We call *connection* the set of SAs that were stored in a party's memory after an IKEv2 phase 1 session (one IKE_SA_INIT exchange and one IKE_AUTH exchange). If everything went well, the term connection thus denotes the set containing one IKE SA and one of its Child SAs. However, if e.g. authentication succeeded but traffic selector negotiation failed, then the term connection denotes the IKE SA that was stored alone. Recall that an IKE SA, like a Child SA, consists of two unidirectional SAs. Hence for a party, if authentication succeeded but traffic selector negotiation failed, installing a connection requires to store two unidirectional SAs in its SAD. We say that a party *handles* a connection from the moment the party installs the connection (stores it in memory) until the moment the party deletes the connection.

Let L_{app} be the application load, i.e. the amount of memory occupied by IKEv2 in Victim when there are no connection installed. We assume that L_{app} stays the same during the attack, i.e. is constant in time. Let L be the load of Victim, i.e the amount of Victim's memory that is occupied by (1) the application and by (2) connections with machines that are not Probe or Initiator machines. In practice, there could be other machines than Probe or Initiator machines initiating or removing connections during the attack, i.e. in general L is not constant in time. However, we assume for the sake of simplicity that L is constant in time. We also assume that Victim's memory is statically limited, i.e. there is no way for Victim to obtain more memory capacity during the attack. Let C be the memory capacity that was allocated to IKEv2 in Victim. We assume that C was carefully chosen and that we have $C > L$.

We say that a connection in a party's memory is *unintended* when it was not taken into account when memory was allocated to the party. Let m be the amount of memory needed to store all data related to one connection.

We say that a connection in a party's memory is *stale from the beginning*, when the party has received no IKEv2 message for it since it was installed. After some time and

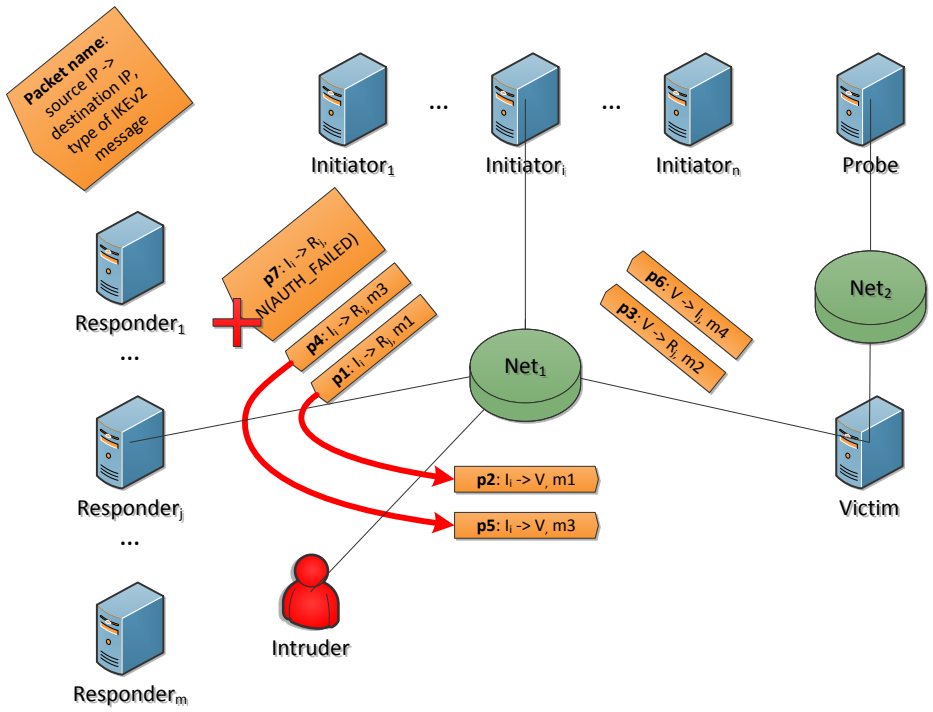


Fig. 6. Scenario of the Deviation Attack. Intruder deviates p1 and p4 and drops p7.

some unanswered rekeying requests or keep-alive requests, the party removes the connection. Let S be the average time a connection stale from the beginning stays in Victim's memory.

Let *Intruder* be a machine, connected to Net_1. Let $t = 0$ be the beginning of the attack and D be the attack duration.

We assume for the sake of simplicity that the Initiator parties send m1 messages to Responder parties at a constant rate σ between $t = 0$ and $t = D$. We express σ in number of IKE messages per second (and not in packets per second, in case there is fragmentation). We also assume that, at $t = 0$, no Initiator party has any SA established with Victim.

For the purpose of detecting an eventual Denial-of-Service, we make Probe regularly send m1 messages to Victim. However, we do not want Probe to affect Victim's memory occupation. We thus make Probe's connections in Victim ephemeral, i.e. Probe's connections are removed after a short time from Victim's memory. See Section V-A for a way to achieve this for the strongSwan IKEv2 implementation.

Let $m4_t$ be the m4 message that Probe receives at time t, and let $m1_t$ be the m1 message sent by Probe that led to $m4_t$ (i.e. that belongs to the same session). Let $T_r(\sigma, t)$ be the response time of Victim to Probe at time t when throughput is σ. We define this response time as the difference between t and the time at which $m1_t$ was sent. We assume that the response time of Victim to Probe is the same as the response time of Victim to any of the $Initiator$ parties. Let T_{acc} be the maximum response time acceptable by Probe, i.e. the response time after which Probe considers that it has been denied a service. The value of T_{acc} is fixed arbitrarily.

We define the following propositions:

Req1 Intruder has the ability to intercept every IP packet sent by an Initiator party to a Responder party. When Intruder intercepts a packet, the recipient does not receive it. In addition, Intruder can either drop the packet, or modify its destination IP address and send it to Victim.

Req2 All Initiator parties authenticate themselves using signature mode.

Req3 All Initiator parties are trusted by Victim. This means that Victim's Peer Authorization Database (see [1]) allows connections with all Initiator parties.

Req4 All m1 messages sent by Initiator parties contain at least one SA proposal (see [1]) that is acceptable to Victim.

B. Attack flow

Assume that $Req1$, $Req2$, $Req3$ and $Req4$ are satisfied. The attack proceeds as follows. Intruder intercepts all m1 messages sent by Initiator parties to Responder parties. Let us consider the implications of one specific m1 message sent by Initiator_i to Responder_j. This message is sent in an IP packet *p1*. The message flow is shown on figure 6.

Thanks to $Req1$, Intruder intercepts the request, changes the destination IP address to V, and sends it to Victim (packet *p2*). We call this process deviation.

In response, because $Req4$ is satisfied, Victim sends an m2 message to Initiator_i (packet *p3*). Initiator_i receives it, and sends an m3 message to Responder_j (packet *p4*). Intruder deviates the m3 message to Victim (packet *p5*).

On reception of the m3 message, authentication of Initiator$_i$ to Victim succeeds because authentication is done using signature mode ($Req2$) and because Initiator$_i$ is trusted by Victim ($Req3$). However, TS and cryptographic algorithms negotiation may fail (TS negotiation will fail in most cases). If TS and cryptographic algorithms negotiation fail, then only one IKE SA is stored. If TS and cryptographic algorithms negotiation do not fail, then one IKE SA and one Child SA are stored. According to our definition of "connection", at this point Victim has installed one connection with Initiator$_i$. Victim then sends an m4 message to Initiator$_i$ (packet *p6*).

On reception of the m4 message, Initiator$_i$ fails the authentication step, since it intended to speak with Responder$_j$, not with Victim. Initiator$_i$ thus sends an IKEv2 notification AUTHENTICATION_FAILED to Responder$_j$ (packet *p7*). Intruder intercepts the notification and drops it (thanks to $Req1$). As a result, an unintended connection was added in Victim's memory.

In this paper, we only explore memory exhaustion as a possible cause of Denial-of-Service. We state that if $Req1$, $Req2$, $Req3$ and $Req4$ are satisfied, then there is a throughput that allows Intruder to cause a Denial-of-Service by memory exhaustion. More formally:

Theorem IV.1. *If $Req1$, $Req2$, $Req3$ and $Req4$ are satisfied, then:*

$$\{\sigma \mid \exists t \in [0, D] \mid T_r(\sigma, t) > T_{acc}\} \neq \emptyset \quad (1)$$

Proof:

Since Victim's memory is statically limited, there exists a throughput σ such that, even if Victim processes the messages infinitely fast, at some time during the attack, Victim is in Denial-of-Service.

■

C. Minimum throughput and DoS time

We can now define Σ_{mem} as the minimum throughput triggering a memory exhaustion.

Definition IV.1.

$$\Sigma_{mem} = min(\{\sigma \mid \exists t \in [0, D] \mid T_r(\sigma, t) > T_{acc}\})$$

When $\sigma > \Sigma_{mem}$, we define $T^s_{mem}(\sigma)$ as the time at which the DoS starts, and $T^e_{mem}(\sigma)$ as the time at which the DoS ends.

Definition IV.2. Let $\sigma > \Sigma_{mem}$. We define:

$$T^s_{mem}(\sigma) = min(\{t \in [0, D] \mid T_r(\sigma, t) > T_{acc}\})$$
$$T^e_{mem}(\sigma) = max(\{t \in [0, D] \mid T_r(\sigma, t) > T_{acc}\})$$

We give a deductive proof of the following theorem in Appendix C. Furthermore we verify the following theorem using experiment in Section V, by implementing the Deviation Attack and measuring DoS times in different experimental setups.

Theorem IV.2 (Memory exhaustion)**.** *We state that:*

$$\Sigma_{mem} = \frac{C - L}{m \times min(D, S)}$$

Furthermore, when $\sigma > \Sigma_{mem}$, memory exhaustion starts and ends at:

$$T^s_{mem}(\sigma) = \frac{C - L}{m\sigma}$$
$$T^e_{mem}(\sigma) = max(D, S)$$

Remark. To give a numerical application of our expressions of Σ_{mem} and T^s_{mem}, we provide in Appendix D a use case of the Deviation Attack where we imagine an attack against a commercial VPN service.

D. Discussion of the Deviation Attack

a) The Deviation Attack when pre-shared keys are in use: Note that we do not require that Victim authenticates itself using signature mode. In other words, even if authentication is performed asymmetrically, with Victim using signature and the Initiator machines using PSK, the attack still works.

We said in Section III-D that IKEv2-PSK does not satisfy weak agreement when A and B share the same PSK as A and C. Therefore, the Deviation Attack is also possible when the Initiator machines and Victim share the same PSK as the Initiator machines and the Responder machines; otherwise it does not work.

b) Why Initiator$_i$ does not refuse message p3: In the context of a Deviation Attack, Initiator$_i$ sees that the source IP address of message p3 (see figure 6) is not the destination address of message p1. At first glance, this observation could be the witness of an odd situation. However, the IKEv2 RFC specifically says that "Incoming IKE packets are mapped to an IKE SA only using the packet's SPI, not using (for example) the source IP address of the packet" [2]. This is why Initiator$_i$ does not refuse message p3.

c) A way to obtain enough requests to deviate: For the attack to work, there need to be a sufficient rate of IKE_SA_INIT requests that are sent from the Initiator parties to the Responder parties, in a duration short enough. If this situation never arises, Intruder may have a workaround: it can drop all messages coming from the Initiator parties and going to the Responder parties, for a given time. After some unanswered IKEv2 keep-alive requests (if Dead Peer Detection is activated, see Section VI-A) or some unanswered CREATE_CHILD_SA requests, the Initiator parties may consider their connections with the Responder parties as broken and may send new IKE_SA_INIT requests for each broken connection. This solution works for example if the connections are configured so as to be automatically set back up when broken (option "closeaction=restart" in strongSwan), or so as to be automatically set up when an outbound IP packet arrives (option "auto=route" in strongSwan).

d) Classification among DoS attacks: The Deviation Attack belongs to the category of slow DoS attacks (SDA). A definition of SDAs is given in [24]. An SDA is a DoS attack that requires very low amount of bandwidth. To do so, SDAs usually target a listening daemon on a host by exploiting some

application-layer vulnerability. Indeed, the Deviation Attack makes an IKEv2 daemon unavailable by exploiting a weakness in the application and requires very low amount of bandwidth compared to classic flooding DoS techniques.

V. Attacking an IKEv2 implementation

To concretely demonstrate the Deviation Attack implications, we attack the strongSwan open-source IKEv2 implementation. Our experiment code is available at [25]. Our experiment also allows us to experimentally verify our expressions of Σ_{mem} and T^s_{mem} that we give in theorem IV.2.

A. Setup

To reproduce the attack, we create 3 Virtual Machines (VMs) representing Victim, Probe and Intruder, and N_{demo} VMs representing $(\text{Initiator}_i)_{1\leq i\leq N}$, where N_{demo} is a configurable parameter. We connect all VMs through a Local Area Network (LAN). See Appendix E for an explanation of why we choose this configuration and how it impacts the machines' configuration.

We use Virtualbox 5.0.40_Ubuntu r115130 for the virtualization, and Vagrant 2.0.0 as a VM managing tool. All machines are under Ubuntu 16.04, with a x86_64 architecture. The host machine has a 2.6 GHz processor with 4 cores. We define $software$ as the IKEv2 implementation we are attacking. In our experiment, $software$ is strongSwan version "Linux strongSwan U5.1.2/K3.13.0-132-generic". Throughout this Section, we use the $software$ term to indicate something that is not specific to strongSwan, but rather would have been true for any IKEv2 implementation. Except when indicated, we use the default options for $software$.

We create a Certificate Authority that we call *CA*. We generate for Probe, Victim and each Initiator VM a certificate signed by CA, and its associated private key. We place in Probe, Victim and each Initiator VM their respective certificates and private keys, along with CA's certificate.

We use Linux Control groups (Cgroups) to allocate a (configurable) memory of exactly C to Victim for $software$. We use option $uniqueids = never$ in Victim, as explained in Appendix E. In Victim we futher disable the Out-Of-Memory killer of the kernel and of $software$'s control group, as explained in Appendix E.

The Probe VM tries to set up a new IPsec connection every 2 seconds. For each attempt, after T_{acc} seconds (configurable), Probe checks if the attempt has succeeded and reports the result. To make Probe's connections in Victim ephemeral as explained in Section IV-A, for strongSwan we use the following options in Victim:

Listing 4. Making Probe's connections in Victim ephemeral

```
# ipsec.conf
conn other
    inactivity=5s
# strongswan.conf
charon.inactivity_close_ike=yes
```

The Initiator VMs try to establish connections with a $Responder$ machine (that we did not instantiate, see Appendix E) at a configurable rate σ. TS payloads sent the Initiator VMs are not valid propositions for Victim. As explained in Section V-A, a connection will thus consist of only one IKE SA (containing two unidirectional SAs). For strongSwan, we set the $rightid$ option in Initiator like below. The "%" sign forces strongSwan not to send IDr in the IKE_AUTH request. We explain in Section V-C why this is necessary.

Listing 5. The rightid option in strongSwan

```
rightid="%CN=responder"
```

In Intruder, we use the $arpspoof$ tool [26] to intercept the requests from Initiator, and the $iptables$ tool to redirect them towards Victim. We stop the attack after some configurable time D.

We thoroughly discuss our setup in Appendix E. In particular, we review all simplifications we make compared to the generic scenario presented in Section IV-A, and explain why they do not prevent the experiment to confirm the attack existence.

B. Strategy for experimentally verifying theorem IV.2

a) Measuring T^s_{mem} is enough to verify theorem IV.2: In our experiments, we experimentally verify our expressions of Σ_{mem} and T^s_{mem} by only measuring T^s_{mem}, and not Σ_{mem}. The following theorem, which we prove in Appendix C, explains why this is a valid approach.

Theorem V.1 (Equivalence theorem)**.** *Experimentally verifying our expression of T_{mem} is sufficient to experimentally verify our expression of T_{mem} and Σ_{mem}.*

b) Measuring T^s_{mem}: To experimentally verify our expression of T_{mem}, we measure T^s_{mem} ($measured\ T^s_{mem}$) when tuple (C, σ) takes its values in $domain$, where $domain$ is defined as follows:

$$domain = \{(50,1),(50,5),(50,10),(50,30),\\ (200,1),(200,5),(200,10),(200,30)\}$$

We measure σ during the experiment ($measured\ \sigma$) and observe that it is different from the σ we configured ($configured\ \sigma$). This is most probably due to virtualization. For the experimental verification of theorem IV.2 not to be affected by the fact that $configured\ \sigma$ and $measured\ \sigma$ are different we thus use $measured\ \sigma$ to calculate the T^s_{mem} value predicted by theorem IV.2.

For consistency we use a warm up run. Furthermore for each element of $domain$ we perform the measure of T^s_{mem} (resp. σ) 10 times. We then take the average of T^s_{mem}'s (resp. σ's) measures as our value of $measured\ T^s_{mem}$ (resp. $measured\ \sigma$).

For each element of $domain$ we use the following expression to calculate the T^s_{mem} value predicted by theorem IV.2:

$$expected\ T^s_{mem} = \frac{C - L}{m \times measured\ \sigma}$$

c) Measuring constants: In order to confront theorem IV.2 with the experiment, we first need to measure constants L and m for $software$ in the context of our setup. We also need to verify that L and m are indeed constants, i.e. that they do not depend on C or σ.

For practical reasons we only verify that m and L are constants when tuple (C, σ) takes its values in $domain$. Since $domain$ only consists of 8 elements, we simply measure m and L on each element and verify that these measures are approximatively the same. If indeed they are approximatively the same, we then take the average of m's (resp. L's) measures as our value of constant m (resp. L).

To measure m we fill Victim's memory with a high number of connections $measuremnumconns$ at a throughput $measuremthroughput$, and divide the memory increase by $measuremnumconns$. There are several ways to measure the amount of memory used by a process. To be consistent with how we limit memory available to $software$ (using Cgroups), we take the value stored in file "/sys/fs/cgroup/memory/$software$/memory.usage_in_bytes".

In our experiment, we have $L = L_{app}$, where L_{app} is the amount of memory occupied by IKEv2 in Victim when there is no connection installed (as defined in Section IV-A). To measure L we thus simply measure the amount of memory used by $software$ when there are no connections installed.

C. Results

a) Vulnerability of strongSwan: We observe that strongSwan differs from IKEv2's RFC on one point. When the IKE_AUTH request IDr payload does not correspond to any of the responder's identities, strongSwan notifies that no matching peer configuration has been found and cancels the IKE SA establishment. In the RFC, it is said the following: "If the IDr proposed by the initiator is not acceptable to the responder, the responder might use some other IDr to finish the exchange". In other words, if IDr does not correspond to one of its identities, the responder might install a Child SA anyway, and use some other IDr to finish the exchange.

The IKE_AUTH request IDr payload is optional, both in the RFC and in strongSwan. The behaviour adopted by strongSwan and described above thus implies the following for the Deviation attack: When the Initiator machines run strongSwan and are configured so as to not send an IDr payload, the attack works. However, when they run strongSwan and are configured so as to send an IDr payload, the attack does not work, since IDr would be equal to a Responder machine's identity, and not Victim's.

Not sending IDr payload in an IKE_AUTH request is not an uncommon configuration, since it allows to hide the responder's identity to an active attacker. We explain this in section VI-A.

b) Experimentally verifying theorem IV.2: Our measures of L and m confirm that L and m do not depend on C or σ (or very little). We obtain $L \approx 1\,\mathrm{MB}$ and $m \approx 18.805\,\mathrm{kB}$.

The results of our measures of T^s_{mem} are shown in table 7. Our measures are close to the values predicted by theorem IV.2: we obtain an average relative error of 1.3% and a maximum relative error of 2.4%. Because of theorem V.1 we thus experimentally verified theorem IV.2.

VI. Counter-measures

A. Trying to protect implementations of the current protocol

Users who cannot afford to be targeted by the Deviation Attack need immediate protection. In this Section, we show that the cookie and puzzle mechanisms that were introduced in IKEv2 to resist DoS attacks are of no use against the Deviation Attack. We then note that Dead Peer Detection can be a small mitigation and consider two measures that prevent the attack but possess significant drawbacks: using PSK authentication and giving enough resources to Victim.

a) Existing DoS counter-measures: The cookie mechanism [2] was introduced to protect IKEv2 against a memory exhaustion due to reception of a large amount of IKE_SA_INIT requests. If this mechanism is in place, when the responder detects a large number of half-open IKE SAs, it responds to each IKE_SA_INIT request (that does not contain a cookie) with an IKEv2 INFORMATIONAL message containing a cookie. The cookie is a keyed hash of the request. The initiator then sends the same request again with the cookie added to it, and the responder verifies that the cookie and the request match. This means that the attacker needs to keep in memory the IKE_SA_INIT requests it sends. It thus makes it more costly for an attacker to fill the half-open SA database of a gateway. However, in the Deviation Attack, cookies will be handled by the Initiator parties and not by Intruder. Activating cookies thus has absolutely no effect on Intruder's memory requirements. Therefore, it does not increase the cost of the attack in terms of memory. Note that the cookie mechanism also increases the time between the reception of an IKE_SA_INIT request by Victim and the filling of its memory with the new SA. But this neither increases the throughput of packets the attacker needs to deviate, nor the duration needed for the Deviation Attack to succeed.

The puzzle mechanism is specified in [27]. It is an improvement of the cookie mechanism. The initiator now needs to remember its request and to solve a puzzle before sending its request again. Like the cookie mechanism, the puzzle mechanism is stateless for the responder. Puzzles increase the cost of an attack in CPU power. However, in the deviation attack, it is the Initiator parties who need to solve the puzzles, not the Intruder. So puzzles, like cookies, are of no use against the Deviation Attack.

b) Dead Peer Detection: Dead Peer Detection [28] (DPD) is a mechanism left as an option in IKEv2. In fact, in strongSwan, it is not enabled by default. When DPD is in use, whenever a party sees that no traffic has recently been received on an IKE SA or any of its Child SAs, then it may send a keep-alive request to the SA's peer. If the peer does not respond, after several retransmissions, the party may remove the IKE SA and all its Child SAs from its memory. In the context of the Deviation Attack, DPD reduces S, the average time a connection stale from the beginning stays in memory. Since Σ_{mem} is inversely proportional to S, DPD makes it harder to achieve a memory exhaustion using the Deviation Attack.

C (in MB)	50	50	50	50	200	200	200	200
Configured σ (in m1 messages/s)	1	5	10	30	1	5	10	30
Measured σ (in m1 messages/s)	1.0	4.8	9.5	25.6	1.0	4.9	9.5	24.1
Expected T^s_{mem} (in s)	2605	541	274	101	10582	2159	1111	438
Measured T^s_{mem} (in s)	2617	547	280	103	10771	2202	1124	445
Relative error in %	0.5	1.1	2.1	2.4	1.8	2.0	1.1	1.6

Fig. 7. Predicting and measuring T^s_{mem} during some Deviation Attacks against strongSwan.

However, reducing S too much can create an overload on the network, so S cannot be too low. When DPD is activated in strongSwan, using default values for the other options, we have $S = 195s$, which multiplies Σ_{mem} by 18. Therefore, DPD only mitigates memory exhaustion.

c) Pre-Shared Key authentication: As we point out in Sections B and IV-B, the Deviation Attack is not possible when PSKs are used for authentication. Using PSK thus is a counter-measure, provided that the Initiator machines and Victim do not share the same PSK as the Initiator machines and the Responder machines. Of course, PSKs and certificates do not fulfil the exact same needs. PSKs are used when the number of peers is relatively low. If this is not the case, another counter-measure has to be considered.

d) Giving enough resources to Victim: One solution to the memory exhaustion is to give enough memory power to Victim to handle as many connections as there can be. Victim would then need to be given a memory of at least $L_{app} + N_t \times m$, where N_t is the number of peers that Victim trusts. In the context of our use case (see Section D), this represents a memory of: $C = L_{app} + 10^8 \times m = 1.9\,\text{TB}$. This counter-measure is efficient since memory is cheap nowadays. However, IKEv2 should not rely on such a recommendation to its users.

B. Improving the protocol specification

Attempts to protect implementations of the current protocol are either not sufficient to prevent the Deviation Attack, or present significant drawbacks. For this reason, we propose two modifications of the protocol specification that we prove deter its vulnerability to the Deviation Attack.

a) Using IDr payload: A way to modify the protocol would be to make the IDr payload in IKE_AUTH request mandatory, and to modify its processing by the responder. The appropriate behaviour would be not to install a Child SA when IDr is not *acceptable* (defined below), but instead, to cancel the establishment of the IKE SA, by sending an AUTHENTICATION_FAILED notification to the Initiator. The message sequence chart on figure 8 shows the modified protocol.

Let us define the term *acceptable* we used above. The IPsec and IKEv2 specifications should have a new mandatory field in the PAD. We call this field the *local ID* field. We should also rename the ID field, described in RFC 4301, Section 4.4.3, to *remote ID field*, for consistency. The local ID field would be a non-empty list of IDs. Each ID would be in the same format as the ID field (it can use wildcards, for example). When an IKE_AUTH request arrives, a PAD lookup is done.

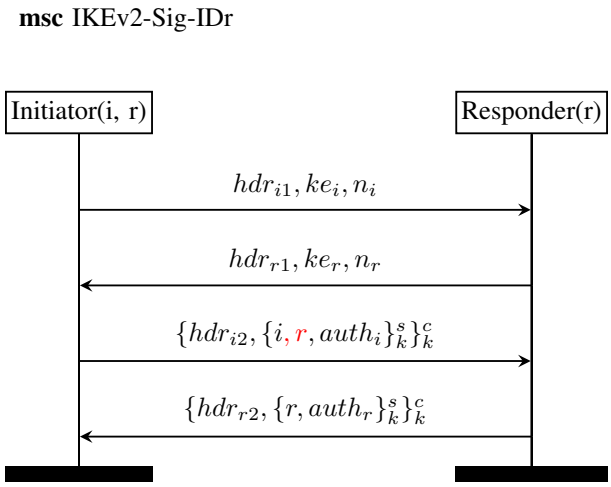


Fig. 8. The IKEv2-Sig-IDr protocol. We make IDr (r in the figure above) payload mandatory in the IKE_AUTH request and modify its processing by Responder.

A PAD entry would match the request when both the remote ID field and the local ID field match respectively the IDi and IDr payloads.

Note that, as we explain in Section V-C, strongSwan already implemented the local ID field. It corresponds to the $leftid$ attribute of the ipsec.conf configuration file.

The IDr modification, however, has one drawback. Assume Alice initiates an IKEv2 session with Bob, using the non-modified protocol. According to [29], IKEv2 was designed so as to hide both identities from a passive attacker and Bob's identity from an active attacker as well. It does not hide Alice's identity from an active attacker because it is Alice who reveals its identity first using IDi payload. The attacker can learn this identity by impersonating Bob's IP address.

Now assume we modify the protocol using the IDr method. Using the same attack where the intruder impersonates Bob, the intruder is now able to learn the identity of Bob, and even to prove that Alice intended to speak to Bob. In other words, this modification works but the responder's identity would no longer be hidden from an active attacker. This may be a problem if sensitive information can be found in the ID payload. This is often the case, as people often use Distinguished Names (DN), where the country, institution name and email address are given.

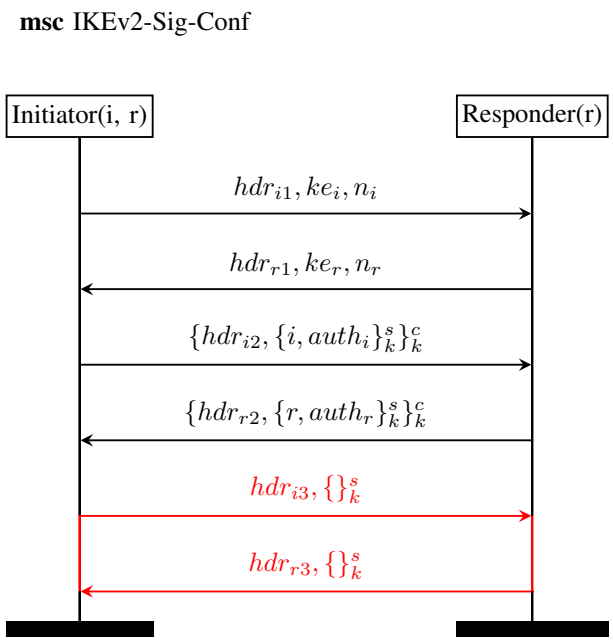


Fig. 9. The IKEv2-Sig-conf protocol. We add the key confirmation exchange. Connections are installed only after this exchange.

b) Adding key confirmation: There is a way to modify IKEv2-Sig that does not require to disclose the responder ID before it is cryptographically verified. The message sequence chart on figure 9 shows the modified protocol.

We add a third exchange, called the *key confirmation* and written KEY_CONF. This exchange is exactly the same as an IKEv2 keep-alive exchange: an empty INFORMATIONAL request and an empty INFORMATIONAL response. We require that the responder installs its connection only after having received a valid KEY_CONF request and that the initiator installs its connection only after having received a valid KEY_CONF response. Key confirmation was first proposed by Basin et al. in [7] as a counter-measure to the penultimate authentication flaw. Since the Deviation Attack exploits the latter, key confirmation is a counter-measure to it.

One other modification could have been to simply add a mandatory AUTH_SUCCESS message as an acknowledgement to the IKE_AUTH response. However, in IKEv2, all messages, except for error messages, exist in pairs. This is because IKEv2 is carried over UDP, so the only way to be sure that a request has been received is to wait for a response message and to set up retransmissions in case it does not arrive (until a timeout). With the KEY_CONF response, the initiator is now sure, when it installs its connection, that the responder has installed its.

Finally, adding an exchange to the protocol can be seen as a increase of its cost, as it will take a more time to establish a connection. But a KEY_CONF message requires only a very little amount of time and computational power to generate and process, because there is no asymmetric cryptography or key derivation operations to perform. It is therefore a very efficient solution to prevent the Deviation Attack.

c) Verification: To verify that the modifications we propose prevent the Deviation Attack, 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 loose any guarantee. Recall that IKEv2-PSK is not vulnerable to the Deviation Attack. 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, preventing the Deviation Attack. The modified Promela models can be found with our other models at [19].

VII. Conclusion and Future Work

In this paper, we have performed a formal analysis of the IKEv2 specification using the Spin model checker. Our analysis refutes the reflection attack that was found before and confirms the penultimate authentication flaw. The latter implies that IKEv2 only satisfies a weak form of authentication. This vulnerability was considered harmless, but we showed that it is not. We designed a novel slow Denial-of-Service attack that exploits it: the Deviation attack. We explained its working flow, and precisely evaluated its requirements and consequences. To concretely demonstrate the attack, we successfully implemented it against the strongSwan open-source implementation. The source code to reproduce the Deviation Attack is available at [25].

The central position that IKEv2 occupies in modern infrastructures leaves no doubt that counter-measures need to be taken. We discussed the efficiency of the available means to protect implementations of the current protocol. However, none of them were complete solutions. Worse, the cookie and puzzle mechanisms that were introduced in IKEv2 to counter DoS attacks are completely ineffective against the Deviation Attack. We thus tackled the problem at a higher level: We proposed two possible inexpensive modifications of the protocol, and formally proved, using model checking, that each of them eliminates the attack.

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.

Finally, as we have seen, this paper outlines the importance of formal authentication properties like aliveness and weak agreement for authentication protocols. Their violation does not necessarily imply a violation of secrecy, but we have shown that it can allow other attacks. In particular, when the protocol sets up some connection in the parties' memories, it can lead to a DoS attack. It could be interesting to verify weak agreement for TLS, SSH, and other stateful authentication protocols.

ACKNOWLEDGMENT

The authors would like to thank Thomas Given-Wilson for its help with writing, and Youcef Ech-Chergui for its IKEv2 expertise. In addition, the authors would like to thank the ANSSI [1] for their technical review of the paper.

REFERENCES

[1] K. S. and S. K., "Security Architecture for the Internet Protocol," Internet Requests for Comments, RFC Editor, RFC 4301, December 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4301.txt
[2] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (IKEv2)," Internet Requests for Comments, RFC Editor, RFC 7296, November 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8019.txt
[3] https://en.wikipedia.org/wiki/ARP_spoofing, 2017, [Online; accessed 11-February-2018].
[4] Google, https://github.com/google/oss-fuzz.
[5] 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 (Cat. No.99CB36344)*, 1999, pp. 216–231.
[6] C. Cremers, "Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 315–334.
[7] A. Project, "Deliverable d6.2: Specification of the problems in the high-level specification language," Tech. Rep., 2003, "http://www.avispa-project.org/".
[8] G. J. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.
[9] 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, pp. 77–86.
[10] "Linear temporal logic," http://spinroot.com/spin/Man/ltl.html, September 2017.
[11] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *J. ACM*, vol. 65, no. 1, pp. 1:1–1:41, Oct. 2017. [Online]. Available: http://doi.acm.org/10.1145/3127586
[12] R. Gerth, "Concise promela reference," http://spinroot.com/spin/Man/Quick.html, June 1997, [Online; accessed 07-March-2018].
[13] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of model checking*. MIT press, 2008.
[14] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.
[15] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, Mar 1983.
[16] D. Basin, C. Cremers, and C. Meadows, "Model checking security protocols," *Handbook of Model Checking*, 2015.
[17] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001.*, 2001, pp. 82–96.
[18] C. J. F. Cremers, *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven, Netherlands, 2006.
[19] https://gitlab.com/deviation/spin.
[20] 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, pp. 340–356.
[21] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 147–166.
[22] ——, "A hierarchy of authentication specifications," in *Proceedings 10th Computer Security Foundations Workshop*, Jun 1997, pp. 31–43.
[23] C. J. Cremers, S. Mauw, and E. P. de Vink, "Injective synchronisation: an extension of the authentication hierarchy," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 139–161, 2006.
[24] E. Cambiaso, G. Papaleo, and M. Aiello, "Slowdroid: Turning a smartphone into a mobile attack vector," in *2014 International Conference on Future Internet of Things and Cloud*, Aug 2014, pp. 405–410.
[25] https://gitlab.com/deviation/demo.
[26] D. Song, https://linux.die.net/man/8/arpspoof.
[27] Y. Nir and V. Smyslov, "Protecting internet key exchange protocol version 2 (ikev2) implementations from distributed denial-of-service attacks," Internet Requests for Comments, RFC Editor, RFC 8019, November 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8019.txt
[28] G. Huang, S. Beaulieu, and D. Rochefort, "A traffic-based method of detecting dead internet key exchange (ike) peers," Internet Requests for Comments, RFC Editor, RFC 3706, February 2004, http://www.rfc-editor.org/rfc/rfc3706.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3706.txt
[29] R. Perlman, "Understanding ikev2: Tutorial, and rationale for decisions," *RFC Editor*, January 2003.
[30] W. contributors, "Arp spoofing," https://en.wikipedia.org/wiki/ARP_spoofing, 2018, [Online; accessed 23-January-2018].

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

APPENDIX A
MESSAGE GENERATION BY THE INTRUDER

The code below is the Promela code of our macro $Randm1m2message$. This macro is used by the intruder to forge new messages, on the basis of its knowledge. The messages are then injected on the network.

Listing 6. The Randm1m2message inline function makes the intruder forge an IKE_SA_INIT message

```
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;
```

msc Penultimate authentication flaw

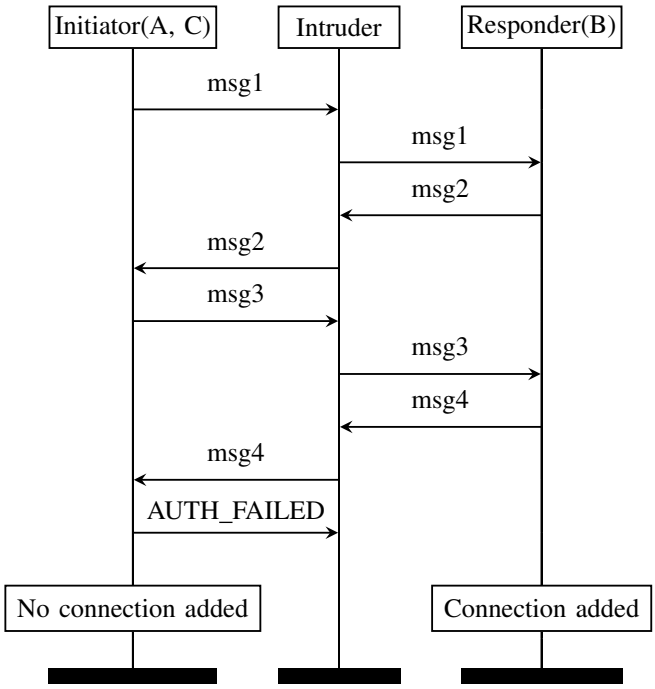


Fig. 10. The penultimate authentication flaw. Violation of weak agreement for B: A wanted to talk to C, not B. IKEv2 messages msg1, msg2, msg3 and msg4 are respectively an IKE_SA_INIT request and its response and an IKE_AUTH request and its response. They are the same before and after deviation.

```
        p11 = NULL;
        p12 = NULL;
        p13 = NULL;
}
```

APPENDIX B
PENULTIMATE AUTHENTICATION FLAW

The penultimate authentication flaw is a weakness of IKEv2-Sig (not IKEv2-PSK). The attack trace is shown on figure 10. It means that IKEv2-Sig satisfies only a weak form of authentication. It was first reported for IKEv2 in [7]. In this attack, A starts a session as initiator and wants to talk to C. But the intruder deviates every message A sends, to Responder B, and every message B sends, back to A (of course messages sent by B were already addressed to A, so whether the intruder does not do anything or intercept it and forward it, the attack remains). The parties proceed normally until A receives the IKE_AUTH response. The AUTH payload is not signed with the private key of C, so A does not set up a Child SA. A then sends an IKEv2 INFORMATIONAL message containing an AUTHENTICATION_FAILED notification payload. Intruder intercepts it and drops it. In the end, B has set up a Child SA with A, whereas A did not want to set up a Child SA with B. This is a violation of weak agreement for the responder.

The reason why IKEv2-PSK is not vulnerable is because, in theory, A and B do not share the same PSK as A and C. However, in practice, PSKs are sometimes shared among several machine pairs, such as between all IKEv2 servers and clients of a company's VPN access infrastructure. In those cases, weak agreement is violated as well.

APPENDIX C
PROOFS OF THEOREMS IV.2 AND V.1

Theorem IV.2 (Memory exhaustion)**.** *We state that:*

$$\Sigma_{mem} = \frac{C - L}{m \times min(D, S)}$$

Furthermore, when $\sigma > \Sigma_{mem}$, memory exhaustion starts and ends at:

$$T^s_{mem}(\sigma) = \frac{C - L}{m\sigma}$$
$$T^e_{mem}(\sigma) = max(D, S)$$

Proof:

Let $\lambda(t)$ be the predicate "Victim suffers from memory exhaustion at time t" and let Λ be the predicate "The attack leads to a memory exhaustion".

We further define the three following predicates:

$$\begin{aligned} expr(\lambda) &\equiv (\lambda(t) \Leftrightarrow \\ &\qquad \sigma \times min(t, S) > \frac{C - L}{m}) \\ expr(T_{mem}) &\equiv (T^s_{mem}(\sigma) = \frac{C - L}{m\sigma}) \\ expr(\Sigma_{mem}) &\equiv (\Sigma_{mem} = \frac{C - L}{m \times min(D, S)}) \end{aligned} \tag{2}$$

Let us prove that $expr(\lambda)$ is true and that:

$$expr(\lambda) \Rightarrow expr(T_{mem})$$
$$expr(\lambda) \Rightarrow expr(\Sigma_{mem})$$

Summing up implications of all m1 messages sent by Initiator parties to Responder parties, at time t, Victim has installed $\sigma \times t$ connections in its memory. However, all these connections are stale from the beginning, so they are removed after S seconds. Therefore at time t, the number of unintended connections that are present in Victim's memory is $\sigma \times min(t, S)$. We thus have:

$$\lambda(t) \iff \sigma \times min(t, S) > \frac{C - L}{m}$$

We have proved that $expr(\lambda)$ is true. Let us now suppose that $expr(\lambda)$ it true. We have:

$$\begin{aligned} \lambda(t) &\iff \sigma \times min(t, S) > \frac{C - L}{m} \\ &\iff t > S \wedge \sigma > \frac{C - L}{mS} \vee t < S \wedge t > \frac{C - L}{m\sigma} \end{aligned}$$

Therefore:

$$
\begin{aligned}
\Lambda &\iff \exists t \in [0, D] \mid \lambda(t) \\
&\iff \exists t \in [0, D] \mid (t > S \wedge \sigma > \frac{C - L}{mS} \\
&\qquad \vee t < S \wedge t > \frac{C - L}{m\sigma}) \\
&\iff (\exists t \in [0, D] \mid t > S \wedge \sigma > \frac{C - L}{mS}) \\
&\qquad \vee (\exists t \in [0, D] \mid t < S \wedge t > \frac{C - L}{m\sigma}) \\
&\iff (D > S \wedge \sigma > \frac{C - L}{mS}) \\
&\qquad \vee (\exists t \in [0, min(D, S)] \mid t > \frac{C - L}{m\sigma}) \\
&\iff D > S \wedge \sigma > \frac{C - L}{mS} \vee min(D, S) > \frac{C - L}{m\sigma} \\
&\iff D > S \wedge \sigma > \frac{C - L}{m \times min(D, S)} \\
&\qquad \vee \sigma > \frac{C - L}{m \times min(D, S)} \\
&\iff \sigma > \frac{C - L}{m \times min(D, S)}
\end{aligned}
$$

In consequence, $expr(\Sigma_{mem})$ is true.

Now we assume that $\sigma > \Sigma_{mem}$. Let us calculate $T^s_{mem}(\sigma)$. We have:

$$
\begin{aligned}
\lambda(t) &\iff t > S \wedge \sigma > \frac{C - L}{mS} \vee t < S \wedge t > \frac{C - L}{m\sigma} \\
&\iff t > S \vee t < S \wedge t > \frac{C - L}{m\sigma} \\
&\iff t < S \wedge t > \frac{C - L}{m\sigma} \\
&\iff t > \frac{C - L}{m\sigma}
\end{aligned}
$$

Memory exhaustion will thus start at:

$$T^s_{mem}(\sigma) = \frac{C - L}{m\sigma}$$

In consequence, $expr(T^s_{mem})$ is true. We have proved that:

$$
\begin{aligned}
expr(\lambda) &\Rightarrow expr(T_{mem}) \\
expr(\lambda) &\Rightarrow expr(\Sigma_{mem})
\end{aligned}
$$

The memory exhaustion lasts until connections are removed and the attack has stopped, i.e. until $T^e_{mem}(\sigma) = max(D, S)$. ■

Theorem V.1 (Equivalence theorem)**.** *Experimentally verifying our expression of T_{mem} is sufficient to experimentally verify our expression of T_{mem} and Σ_{mem}.*

Proof:

Let us prove that $expr(T_{mem}) \Rightarrow expr(\lambda_{mem})$. Suppose that $expr(T_{mem})$ is true. We have:

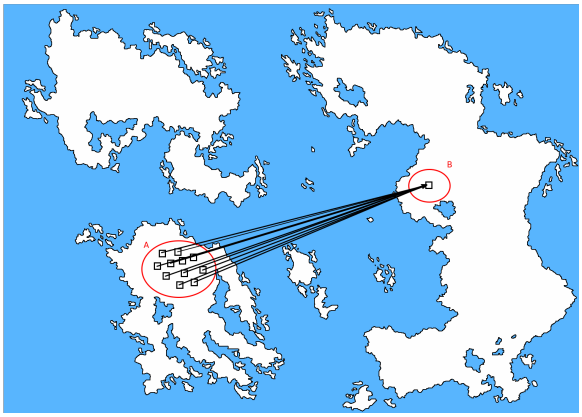


Fig. 11. Use case of a IoT fleet. An intruder deviates all m1 messages coming to servers in country A towards a server in country B.

$$
\begin{aligned}
\lambda(\sigma, t) &\Leftrightarrow t > T^s_{mem}(\sigma) && \text{by definition of } T^s_{mem} \\
&\Leftrightarrow t > \frac{C - L}{m\sigma} && \text{by 2} \\
&\Leftrightarrow \sigma \times min(t, S) > \frac{C - L}{m} && \text{if S is big enough}
\end{aligned}
$$

Therefore, $expr(\lambda_{mem})$ is true. We have proved that $expr(T_{mem}) \Rightarrow expr(\lambda_{mem})$. Using the two implications proved in theorem IV.2's proof, we thus have:

$$
\begin{aligned}
expr(\lambda) &\Leftrightarrow expr(T_{mem}) \\
expr(\lambda) &\Rightarrow expr(\Sigma_{mem})
\end{aligned}
$$

Therefore $expr(T_{mem}) \Rightarrow expr(\lambda_{mem})$ and experimentally verifying $expr(T_{mem})$ is sufficient to experimentally verify both $expr(T_{mem})$ and $expr(\Sigma_{mem})$. ■

APPENDIX D
AN EXAMPLE USE CASE

Let us consider a company that owns a fleet of Internet of Things (IoT) devices. Each device is constantly connected with one of the company's servers through an IPsec connection. Every time the connection is broken, the device sets it back up using IKEv2. Authentication is performed using digital signature. The devices set up connections with their closest server. As they can move, all devices are trusted by all servers.

The company owns 1000 servers and 100 000 000 devices all around the world. Let us consider a country A, where the company possesses 100 servers. Assume the existence of an intruder that can intercept all IKEv2 packets going from and to the servers in this country. Let us also consider a small country B where there is only one VPN server of the company. The intruder has interest in taking down this one server. See figure 11 for an illustration of the use case.

We assume that the company has chosen to allocate statically 64 GB of memory in each server. We also assume that all servers have the same average load. At any time, about 100 000 devices are connected to a server. We assume that none

of the servers use Dead Peer Detection (see Section VI to see the impact of this option). We assume that TS payloads will not be acceptable. In this case a connection thus consist of one IKE SAs (see IV-A). All servers have an IKE SA lifetime of 3h, so a connection stale from the beginning will be removed after 3h.

We take the values L_{app}, m from our measures in Section V-C. We have:

$$\begin{aligned} L &= L_{app} + 100000 \times m \\ &= 1 \times 10^6 + 100000 \times 18.805 \times 10^3 \\ &= 1.9\,\mathrm{GB} \end{aligned}$$

We assume that, whenever the connection is broken, every device tries to reconnect as soon as an IP packet needs to be protected by the connection. This corresponds to the "auto=route" option in the strongSwan implementation. We explain how to exploit this behaviour in a Deviation Attack in Section IV-D. According to what we said above, in average, before $t = 0$, there are 10 000 000 connections in total established with all servers of country A. At $t = 0$, Intruder brakes all connections with servers in A. We assume that within the next hour, all devices that were connected have sent an IKE_SA_INIT request for a new connection to a server in A. In this scenario:

$$\begin{aligned} &C = 64\,\mathrm{GB} && D = 1\,\mathrm{h} \\ &L = 1.9\,\mathrm{GB} && \sigma = \frac{10^7}{3600} = 2778\,\mathrm{m1/s} \\ &m = 18.805\,\mathrm{kB} && S = 3\,\mathrm{h} \end{aligned}$$

Therefore:

$$\Sigma_{mem} = \frac{C - L}{m \times min(D, S)} = 917\,\mathrm{m1/s}$$

We see that the Deviation Attack would lead to a memory exhaustion in Victim. Furthermore, memory exhaustion starts and ends at:

$$\begin{aligned} T^s_{mem} &= min(\frac{C - L}{m\sigma}, S) = 20\,\mathrm{min}\ 49\,\mathrm{s} \\ T^e_{mem}(\sigma) &= max(D, S) = 1\,\mathrm{h} \end{aligned}$$

APPENDIX E
DISCUSSION OF OUR EXPERIMENT SETUP

In our experiment, we made several simplifications compared to the generic scenario presented in Section IV-A. We claim that they do not prevent the experiment to confirm the attack's existence. First, we did not instantiated the Responder parties, since we only need their IP addresses. We do not really need the machines.

Second, we decided to make the Initiator VMs intending only to talk to one Responder peer. This makes it easier to implement Intruder because that way it only needs to spoof one IP address. However in strongSwan, when a party needs to set up a new Child SA with a peer, and already has an IKE SA set up with it, the party will reuse this IKE SA and send a CREATE_CHILD_SA request. This would not be faithful to the generic scenario, so we use the option $reuse\_ikesa = no$ in the Initiator VMs. This tells Initiator to send an IKE_SA_INIT request every time it wants to set up a Child SA.

Third, we decided, for practical reasons, not to create the N Initiator machines of the generic scenario, but instead to only create N_{demo} Initiator VMs, with each of them sending $\frac{N}{N_{demo}}$ m1 messages to Responder machines. However, by default in strongSwan, a party requires an IKE ID to be unique among the IKE SAs it manages. That is, when that party receives an IKE_AUTH request with an IDi payload that is equal to the peer ID of an existing IKE SA in its SAD, it will delete the old IKE SA and replace it with the new one. Because of this in our experiment, the Deviation Attack would add only N_{demo} unintended connections in Victim. To stay faithful to the generic scenario, we thus use the option $uniqueids = never$ on Victim, which tells a party not to delete the old IKE SA in this situation, and instead, to set up the new one alongside. This way, the Deviation Attack adds up to N unintended connections in Victim.

Finally, all machines are connected through the same local (virtualized) network, and they are the only ones connected to it. This ensures more control over networking propagation times and over the daemon's resource loads. Another reason is that it allows us to easily reproduce the deviation of packets by Intruder, because we can use an ARP cache poisoning attack [30]. In this attack, Intruder sends ARP replies to all Initiator VMs, binding its MAC address to Responder's IP. This way, all packets sent by Initiator VMs to Responder are intercepted by Intruder. We then use Linux $iptables$ command to redirect this traffic towards Victim and to drop the AUTH_FAILED notification. Using the notations of Section IV, we have $Net_1 = Net_2 = LAN$. Of course, in reality, when Net_1 is not a LAN, the deviation has to be made using other ways.

We observed in our experiment that when there is no memory left for $software$, the Linux Out-Of-Memory killer (OOM killer) kills the $software$ process. This leads to the loss of all installed SAs. This is undesirable behaviour: setting back up all SAs might take some time, meanwhile suspending the protected IP flow that used traffic SAs. To observe a memory exhaustion in our experiment, we therefore disable the OOM killer.