Formal Analysis of Combinations of Secure Protocols (Extended Version)

Blot Elliott, Jannik Dreier, Pascal Lafourcade

Formal Analysis of Combinations of Secure Protocols*

Elliott Blot[1] and Jannik Dreier[2] and Pascal Lafourcade[1]

1. LIMOS, University Clermont Auvergne, France
2. LORIA, Université de Lorraine, INRIA, CNRS, France

Abstract. When trying to prove the security of a protocol, one usually analyzes the protocol in isolation, i.e., in a network with no other protocols. But in reality, there will be many protocols operating on the same network, maybe even sharing data including keys, and an intruder may use messages of one protocol to break another. We call that a *multi-protocol attack*. In this paper, we try to find such attacks using the Tamarin prover. We analyze both examples that were previously analyzed by hand or using other tools, and find novel attacks.

1 Introduction

When analyzing the security of protocols, one aims at proving specific security properties. The most common types of properties are *secrecy*, meaning that an intruder cannot know a secret value, and *authentication*, meaning that if A thinks he is talking to B, then he is really talking to B. In our digitalized world there are more and more cryptographic protocols everywhere, and we want to verify them to ensure their security.

It is not realistic to assume that a protocol is running alone in the network, and in the real world, an intruder can try to use messages of other protocols in the network to break a protocol. That is what we call a *multi-protocol attack*.

More precisely, we study the following problem of multi-protocols attacks. Given two protocols that ensure a certain security property in isolation, are they still safe for this property if we put them together in the same network? Unsurprisingly there exist many combinations of protocols where this is not the case, i.e., where we can mount multi-protocols attacks.

There are a lot of tools for automatic analysis of security properties, like ProVerif [3], AVISPA [2], Athena [25], Scyther [9], or Tamarin [21]. But they are generally used to analyze the security of a protocol executed in isolation, meaning that each agent only executes the analyzed protocol. In this paper, our aim is also to automatically find multi-protocols attacks using Tamarin.

* This work was supported in part by the project C-ROADS, by the Indo-French Centre for the Promotion of Advanced Research (IFCPAR) and Center Franco-Indien Pour LA Promotion DE LA Recherche Avancee (CEFIPRA) through the project DST/CNRS 2015-03 under DST-INRIA-CNRS Targeted Programme.

Contributions: Several multi-protocols attacks have been found manually or using other tools, our aim is to find them automatically using the Tamarin prover [21]. Our contributions are the following:

- We developed an algorithm to simplify the process of creating the multi-protocol specification file in Tamarin from the individual protocol specifications. Since it is one of the main difficulty of using Tamarin for multi-protocol attacks.
- We analyzed all the protocols given in [10], where the authors also used Scyther a different verification tool. We confirm the results from Scyther using Tamarin.
- We also automatically find all the manual attacks described in [20]. Moreover, we find novel different attacks on the same properties, or unknown attacks on different properties. It demonstrates the limitations of a manual approach. It also confirms that automatic verification is clearly the right approach for security of cryptographic protocols.

Related work: The existence of multi-protocol attack have been introduced by Kelsey, Schneier, and Wagner in [16]. In this paper the attacks were found manually and the authors only consider protocols crafted to break other protocols.

In [20], Mathuria, Raj Singh, Venkata Sharavan, and Kirtankar found six multi-protocol attacks based on 13 protocols from literature: Denning-Sacco protocol [12], amended Woo-Lam protocol [5], ISO Five-Pass protocol [6], Abadi-Needham protocol [1], six protocols from Perrig and Song using APG [24], and three protocols from Zhou and Foley using ASPB [28]. In contrast to these works, we use an automatic verification tool to find these attacks.

Cremers found many multi-protocol attacks in [10], using the tool Scyther, with 30 protocols from literature including Needham Schroeder protocol [23], Needham Schroeder symmetric key protocol [23], Needham Schroeder symmetric key amended protocol [22], Lowe's modified version of the Needham Schroeder protocol [17], SPLICE/AS [27], Hwang and Chen's version of SPLICE/AS [14], Clark and Jacob's version of SPLICE/AS [7], a basic SOPH example (Secret-Out Public-Home), Woo Lam pi f [26], Kao Chow v.1, v.2 and v.3 [15], Yahalom's protocol [4], and Lowe's version of Yahalom protocol [19]. Compared to this work we use the Tamarin instead of Scyther.

Outline: The paper is structured as follows. In Section 2, we present the results we obtain and we compare them with those obtained manually in [20] or using Scyther [10]. Then, Section 3 discusses our workflow in Tamarin, and finally the last section concludes the paper.

2 Multi-Protocols Attack

First we define the properties that we want to verify for each protocol. We define one property for secrecy and two authentication properties.

- *Secrecy* (as define in [8]): if a protagonist A claims the secrecy of a variable N_A at the end of the run of the protocol, then an intruder cannot know this variable.
- *Non-injective agreement* (as define in [11]): if a protagonist A completes a run apparently with B, then B has run the protocol with A and A agrees with all other protagonist on all values. This is not exactly the same definition as in [18], but we keep this definition because it is this one that is used by Scyther.
- *Non-injective synchronisation* [11]: if a protagonist A completes a run as the initiator apparently with B as the responder, then B has run the protocol as the responder with A, and all messages sent and received are exactly like described in the specification of the protocol, in the same order.

We call a type-flaw attack an attack where the intruder uses data of a different type than the data expected by the protocol. For example, in such an attack, the intruder could use two nonces N_1, N_2 instead of another single nonce N ($N = (N_1, N_2)$), or uses an ID as a nonce. We consider separately the case where the intruder can make type-flaw attacks (such attacks are valid) and the case where the intruder cannot (such attacks are not valid).

All our Tamarin files are available online [13].

2.1 Attacks by Cremers [10]

First we study the protocols analyzed in [10] using Scyther. We modeled all these protocols individually in Tamarin. Figure 1 presents our results using Tamarin for the properties described previously, and Figure 2 presents our results for multi-protocols using Tamarin, where we verify the properties for the first of the two protocols. In these figures, ni-synch stands for non-injective synchronisation, sec stands for secrecy and ni-agree stands for non-injective agreement. Moreover, ✓ means that we did not find any attacks, and ✗ means there is at least one attack for the property. A yellow box means that the first protocol (the one for which we verify the security properties in the combination) is safe for this property in isolation, and red box means that both protocols are safe for this property in isolation. An empty box means that the property is not relevant for this protocol. For example, if the key K_{AB} does not exist in the protocol, the properties secrecy A K_{AB} and secrecy B K_{AB} make no sense. Similarly if a protagonist A never obtains a nonce N_B in the property secrecy A N_B is not meaningful.

We confirm the results from Scyther using Tamarin. Note that we did not consider type-flaw attacks for these protocols, as otherwise we get too many trivial attacks as many of the protocols alone become insecure. All timings were measured on a PC with 6 CPUs at 2 Ghz and 32 GB of memory.

We can see in Figure 2 that even if two protocols are safe in isolation for a property, it is not guaranteed that the combination of this two protocols is safe too if they share keys, and multi-protocol attacks are not only due to the other protocol that is not safe for this property.

name	ni-synch A	ni-synch B	sec A N_A	sec B N_A	sec A N_B	sec B N_B	sec A K_{AB}	sec B K_{AB}	ni-agree A	ni-agree B
NSS	0:07 ✓	0:06 ✓	0:01 ✗	-	0:02 ✓	0:01 ✓	0:02 ✓	0:01 ✓	0:03 ✓	0:02 ✓
NSSA	0:16 ✗	0:37 ✗	0:05 ✗	-	0:35 ✓	10:28 ✓	0:03 ✓	0:06 ✓	0:03 ✗	0:02 ✗
NS	0:28 ✓	0:05 ✗	0:01 ✓	0:01 ✗	0:08 ✓	0:02 ✗	-	-	0:12 ✓	0:04 ✗
NSL	0:22 ✓	1:07 ✓	0:01 ✓	0:04 ✓	0:07 ✓	0:02 ✓	-	-	0:08 ✓	0:17 ✓
AS	0:09 ✗	0:04 ✗	0:03 ✗	0:02 ✗	-	0:02 ✗	-	-	0:11 ✗	0:01 ✗
AShc	0:05 ✗	0:05 ✗	0:04 ✗	0:03 ✗	-	0:03 ✗	-	-	0:45 ✗	0:02 ✗
K	0:06 ✗	0:15 ✗	0:00 ✗	0:16 ✗	0:03 ✗	0:01 ✗	0:02 ✓	0:32 ✓	0:07 ✗	0:12 ✗
K2	0:05 ✗	1:40 ✗	0:04 ✗	0:30 ✗	0:04 ✗	0:30 ✗	0:04 ✓	0:21 ✓	0:04 ✗	0:40 ✗
K3	0:03 ✗	5:43 ✗	0:02 ✗	0:02 ✗	0:04 ✗	0:05 ✗	0:05 ✓	2:27 ✓	0:02 ✗	4:26 ✗
WLpif	0:00 ✗	0:01 ✗	-	-	0:00 ✗	0:00 ✗	-	-	0:00 ✗	0:01 ✗
Y	0:04 ✗	5:53 ✗	0:04 ✗	0:12 ✗	0:24 ✓	0:13 ✗	0:09 ✓	0:12 ✗	0:03 ✗	4:18 ✗
YL	0:12 ✓	0:32 ✓	0:01 ✗	0:02 ✗	0:07 ✓	0:15 ✓	0:05 ✓	0:11 ✓	0:06 ✓	0:17 ✓
AScj	0:06 ✗	0:25 ✗	0:05 ✗	0:05 ✓	-	0:01 ✗	-	-	0:02 ✗	0:06 ✗
soph	0:00 ✓	0:01 ✗	0:00 ✗	0:01 ✗	-	-	-	-	0:00 ✓	0:01 ✗

Fig. 1. Results found using Tamarin with NS = Needham Schroeder [23], NSS = Needham Schroeder Symmetric Key [23], NSSA = Needham Schroeder Symmetric Key Amended [22], NSL = Needham Schroeder Lowe [17], AS = SPLICE/AS [27], AShc = Hwang and Chen version of SPLICE/AS [14], AScj = Clark and Jacob version of SPLICE/AS [7], K = Kao Chow [15], K2 = Kao Chow v.2 [15], K3 = Kao Chow v.3 [15], WLpif = Woo Lam pi f [26], Y = Yahalom [4], YL = Yahalom Lowe [19], soph = a SOPH basic example. ni-synch denotes non-injective synchronisation, ni-agree denotes non-injective agreement, and sec A N_A denotes the fact that A claims the secrecy of N_A.

We would expect that Tamarin takes more time to analyze properties for multi-protocols than for protocols in isolation, due to the increased number of transitions and the larger number of traces with the new protocol.

But as we can see in Figure 1 and 2, this is not always the case, like for example for the property ni-synch A for Kao Chow (K) in Figure 1, and for Kao Chow + Woo Lam pi f (K+ WLpif) in Figure 2. This is generally due to the

name	ni-synch A	ni-synch B	sec A N_A	sec B N_A	sec A N_B	sec B N_B	sec A K_{AB}	sec B K_{AB}	ni-agree A	ni-agree B
NSS + NSSA	0:53 ✗ *	1:35 ✗ *	0:12 ✗	-	10:39 ✓	0:32 ✗ **	0:05 ✓	1:34 ✗ **	0:32 ✗ *	0:24 ✗ *
NS + AS	0:40 ✓	0:09 ✗	0:01 ✓	0:02 ✗	0:14 ✗ **	0:03 ✗	-	-	0:16 ✓	0:06 ✗
NS + AShc	0:39 ✓	0:08 ✗	0:01 ✓	0:02 ✗	0:14 ✗ **	0:03 ✗	-	-	0:18 ✓	0:07 ✗
NSL + AS	0:08 ✓	4:45 ✓	0:02 ✓	0:13 ✓	0:08 ✗ **	0:08 ✗ *	-	-	0:04 ✓	0:53 ✓
NSL + AShc	0:28 ✓	18:44 ✓	0:01 ✓	0:21 ✓	0:19 ✗ **	0:14 ✗ *	-	-	0:11 ✓	0:23 ✓
K + WLpif	0:03 ✗	0:11 ✗	0:02 ✗	0:03 ✗	0:02 ✗	0:03 ✗	0:01 ✓	0:03 ✗ **	0:03 ✗	0:06 ✗
K2 + WLpif	0:05 ✗	0:22 ✗	0:03 ✗	0:04 ✗	0:03 ✗	0:03 ✗	0:05 ✓	0:03 ✗ **	0:07 ✗	0:15 ✗
K3 + WLpif	0:03 ✗	0:23 ✗	0:02 ✗	0:32 ✗	0:02 ✗	0:03 ✗	0:11 ✓	0:03 ✗ **	0:02 ✗	0:29 ✗
YL + Y	1:52* ✗	3:12* ✗	1:03 ✗	1:01 ✗	2:21 ✓	19:37 ✓	1:15 ✓	6:48 ✓	1:01 ✗ *	6:43 ✗ *
AScj + soph	0:15 ✗	0:06 ✗	0:11 ✗	0:09 ✗ *	-	0:02 ✗	-	-	0:04 ✗	0:04 ✗

Fig. 2. Result found with Tamarin. NS = Needham Schroeder [23], NSS = Needham Schroeder Symmetric Key [23], NSSA = Needham Schroeder Symmetric Key Amended [22], NSL = Needham Schroeder Lowe [17], AS = SPLICE/AS [27], AShc = Hwang and Chen's version of SPLICE/AS [14], AScj = Clark and Jacob's version of SPLICE/AS [7], K = Kao Chow [15], K2 = Kao Chow v.2 [15], K3 = Kao Chow v.3 [15], WLpif = Woo Lam pi f [26], Y = Yahalom [4], YL = Yahalom Lowe [19], soph = a SOPH basic example. ni-synch denotes non-injective synchronisation, ni-agree denotes non-injective agreement, and sec A N_A denotes fact that A claims the secrecy of N_A. * = the first protocol is safe in isolation, ** = both protocol are safe in isolation

fact that Tamarin finds an attack more rapidly than a proof as Tamarin stops after the first attack it finds (it does not try to find all attacks).

It can also happen that Tamarin proves a property for the combination of protocols more quickly than for the protocols in isolation, like for example Needham Schroeder Lowe in Figure 1 and Needham Schroeder Lowe + SPLICE/AS in Figure 2 for ni-synch A. This can occur for example if the precomputations are the dominating part of the total runtime.

2.2 Attacks by Mathuria et al. [20]

We try to find the attacks described in [20] using Tamarin, to see if we find the same or different attacks if we use an automatic tool. The properties verified are not clearly defined in [20], so we keep the properties as defined previously. More precisely, we verified different authentication properties: non-injective synchronization, non-injective agreement, and a weaker agreement property. The property non-injective agreement as define previously is too strong to get comparable result with the paper, most of protocols of the paper are not safe for this property even in isolation. So we consider a weaker authentication property defined as follows:

- *weaker agreement*: if B thinks that a nonce N_A is generated by A, then A has generated N_A and B authenticates A (called Aut A in Figure 3 and 4)

Figure 3 summarizes results that we obtain with Tamarin in isolation on protocols from [20], and Figure 4 summarizes results we obtain for the multi-protocols. As previously, ni-synch stands for non-injective synchronization, sec stands for secrecy and ni-agree stands for non-injective agreement. Moreover ✓ means that we did not find any attacks, and ✗ means there is at least one attack for the property. A yellow box means that the first protocol (the one for which we verify security in the combination) is safe for this property in isolation, and red box means that both protocols are safe for this property in isolation. An empty box means that the property is not relevant for this protocol.

We can see in Figure 3 in the case of APG.3 for non-injective synchronization and non-injective agreement, all attacks which we found in isolation are type-flaw attacks, and the protocol is safe if we do not consider type-flaw attacks. But attacks we found for APG.3 with APG.2 are not type flaw attacks (see 2.2), so we consider type-flaw attacks separately in this paper. But in the case of ZF.2, we have a protocol that is not safe for any property, considering type-flaws or not. So it is useless to see if ZF.2 can have a multi-protocol attack for a property in combination with an other protocol, a point that the authors of the original paper missed most likely since they searched for attacks manually.

The property weaker agreement seems to be closest to the property used in [20], because we found the same attacks for some protocols. Thus, in rest of the paper, we only present attacks on this property.

In comparison to the original paper we have found, using Tamarin, sometimes different attacks, and sometimes new attacks on the authentication of other protagonists in the same combination of protocols.

In all protocols, we have three participants, A the initiator, B the responder, and S the trusted server. We use symmetric encryption, so S shares the key K_{AS} (respectively K_{BS}) with A (respectively B). Moreover, K_{AB} denotes the session key between A and B, and N_A (respectively N_B) a nonce generated by A (respectively B). Then $\{M\}_K$ denotes the cipher-text obtained by encrypting a message M with the symmetric key K. We assume that each participant shares the same key for both protocols. In the following, when we talk about authentication, we talk about non-injective agreement.

name	ni-synch A	ni-synch B	sec A N_A	sec B N_A	sec A N_B	sec B N_B	sec A K_{AB}	sec B K_{AB}	ni-agree A	ni-agree B	Aut A	Aut B
APG.1	0:14 ✓	0:03 ✓	0:01 ✗	0:01 ✗	0:08 ✗	0:01 ✗	-	-	0:06 ✓	0:01 ✓	0:01 ✓	0:04 ✓
APG.2	0:09 ✓	0:03 ✓	0:01 ✗	0:01 ✗	0:05 ✗	0:01 ✗	-	-	0:05 ✓	0:01 ✓	0:01 ✓	0:02 ✓
APG.3	0:06 ✗ *	0:05 ✗ *	0:01 ✗	0:01 ✗	0:08 ✗	0:02 ✗	-	-	0:05 ✗ *	0:02 ✗ *	0:01 ✓	0:06 ✓
APG.4	0:37 ✓	0:21 ✓	0:04 ✗	0:04 ✗	0:18 ✓	0:01 ✓	0:13 ✓	0:04 ✓	0:23 ✓	0:09 ✓	0:06 ✓	0:06 ✓
APG.5	0:48 ✓	25:57 ✓	0:05 ✗	0:03 ✗	0:20 ✓	2:04 ✓	0:14 ✓	6:18 ✓	0:30 ✓	11:37 ✓	0:29 ✓	0:00 ✓
APG.6	0:04 ✗	0:06 ✗	0:01 ✗	0:02 ✗	0:02 ✗	0:02 ✗	0:06 ✓	0:04 ✓	0:04 ✗	0:02 ✗	0:02 ✓	0:04 ✓
DS	0:01 ✗	0:01 ✗	-	-	-	-	0:01 ✓	0:01 ✓	0:01 ✓	0:01 ✗	0:00 ✓	-
AWL	0:01 ✗	0:01 ✗	-	-	0:00 ✗	0:00 ✗	-	-	0:00 ✗	0:00 ✗	0:00 ✗	0:00 ✓
ISO5	0:19 ✓	0:33 ✓	0:02 ✗	0:03 ✗	0:04 ✓	0:09 ✓	0:01 ✓	0:09 ✓	0:05 ✓	0:16 ✓	0:06 ✓	0:04 ✓
AN	0:01 ✗	0:01 ✗	0:00 ✗	0:01 ✗	-	0:01 ✗	0:01 ✓	0:02 ✓	0:00 ✗	0:00 ✗	0:01 ✗	0:00 ✗
ZF.1	0:16 ✗	5:01 ✗	0:05 ✗	0:24 ✗	0:06 ✗	0:18 ✗	0:13 ✗	1:08 ✓	0:02 ✗	0:39 ✓	0:11 ✓	0:06 ✗
ZF.2	0:01 ✗	0:06 ✗	0:01 ✗	0:01 ✗	0:01 ✗	0:01 ✗	0:01 ✗	0:01 ✗	0:01 ✗	0:04 ✗	0:07 ✗	0:01 ✗
ZF.3	28:25 ✓	1:08 ✗	0:39 ✗	0:12 ✗	0:33 ✗	0:11 ✗	4:50 ✓	0:57 ✗	2:10 ✓	0:22 ✗	0:47 ✓	0:35 ✓

Fig. 3. Results found with Tamarin with APG from [24], DS = Denning Sacco [12], AWL = Amended Woo Lam [5], ISO5 = ISOFive-Pass [6], AN = Abadi Needham [1], ZF from [28], * = only type-flaw attacks.

In the following we discuss our results in details. First we discuss attacks that we found and that differ from those presented in [20], then we present new attacks for properties that were not analyzed in [20].

Different Attacks

APG.4 with APG.6: The first attack is on the authentication of A. In this attack, two protagonists A and A' initiate the *APG.6* [24] protocol with B, and the intruder C pretends to be A in *APG.4* [24]. In the protocol initiated by A', C learns $(N_{A'}, N'_B, A')$, used as a session key, and its ciphertext $\{N_{A'}, N'_B, A'\}_{K_{BS}}$. In the protocol initiated by A, C learns the nonce N_B, used to authenticate to B. In Figure 5, steps on the left hand side are steps of *APG.4*, and steps on the right hand side are steps from *APG.6*.

name	ni-synch A	ni-synch B	sec A N_A	sec B N_A	sec A N_B	sec B N_B	sec A K_{AB}	sec B K_{AB}	ni-agree A	ni-agree B	Aut A	Aut B
APG.1 + APG.2	0:01 ✗ **	0:08 ✗ **	0:01 ✗	0:01 ✗	0:01 ✗	0:02 ✗	-	-	0:01 ✗ **	0:03 ✗ **	0:02 ✗ **	0:01 ✗ **
APG.3 + APG.2	0:01 ✗	0:08 ✗	0:01 ✗	0:03 ✗	0:02 ✗	0:03 ✗	-	-	0:01 ✗	0:03 ✗	0:03 ✗ **	0:01 ✗ **
APG.4 + APG.6	0:18 ✗ **	1:20:54 ✗ *	0:05 ✗	0:06 ✗	4:54 ✓	53:41 ✗ *	2:09 ✓	1:27:40 ✗ **	0:15 ✗ *	1:00:48 ✗ *	4:27 ✗ **	0:28 ✗ **
APG.5 + APG.6	0:08 ✗ **	28:40 ✗ *	0:07 ✗	0:03 ✗	2:05 ✓	0:02 ✗ *	1:23 ✓	0:03 ✗ **	0:04 ✗ *	23:28 ✗ *	4:45 ✗ **	0:07 ✗ **
DS + AWL	0:01 ✗	0:01 ✗	-	-	-	-	0:01 ✗ **	0:01 ✗ **	0:01 ✗ *	0:01 ✗ *	0:01 ✗ *	-
ISO5 + AN	0:03 ✗ *	0:09 ✓	0:01 ✗	0:03 ✗	0:01 ✗ **	0:04 ✓	0:01 ✗ **	0:06 ✗ **	0:02 ✗ *	0:03 ✗ *	0:04 ✓	0:01 ✗ *
ZF.2 + ZF.1	0:01 ✗	0:39 ✗	0:07 ✗	0:12 ✗	0:06 ✗	0:12 ✗	0:05 ✗	0:12 ✗	0:01 ✗	0:33 ✗	3:52 ✗	0:11 ✗
ZF.3 + APG.2	2:48 ✗ **	2:17 ✗	0:08 ✗	1:19 ✗	4:01 ✗	1:20 ✗	0:07 ✗ **	3:52 ✗	59:21 ✗ **	1:34 ✗	1:29:39 ✓	0:38 ✗ **

Fig. 4. Results found with Tamarin with APG from [24], DS = Denning Sacco [12], AWL = Amended Woo Lam [5], ISO5 = ISOFive-Pass [6], AN = Abadi Needham [1], ZF from [28], * = the first protocol is safe in isolation, ** = both protocols are safe in isolation.

This attack is a type-flaw attack, because the intruder uses $(N_{A'}, N'_B, A')$ as a session key. So we blocked type-flaw attacks in Tamarin to see if there are other types of attacks, and we did not find other attacks on the authentication of A.

Denning-Sacco with Amended Woo-Lam: This attack is on the authentication of A. Again, it is a type-flaw attack, because the intruder uses (K_{AB}, T) as a nonce. In this attack, the intruder C plays the role of A and S in both protocols. First, B initiates a protocol *Woo-Lam*[5] with C who impersonates A. Then C sends the ID of A and a fake session key and a timestamp as a nonce. B encrypts that and C has now a valid message to send to B in *Denning-Sacco*[12]. This attack is described in Figure 6.

We did not find other types of attacks for this protocol when we blocked type-flaw attacks in Tamarin using a modified model.

New Attacks

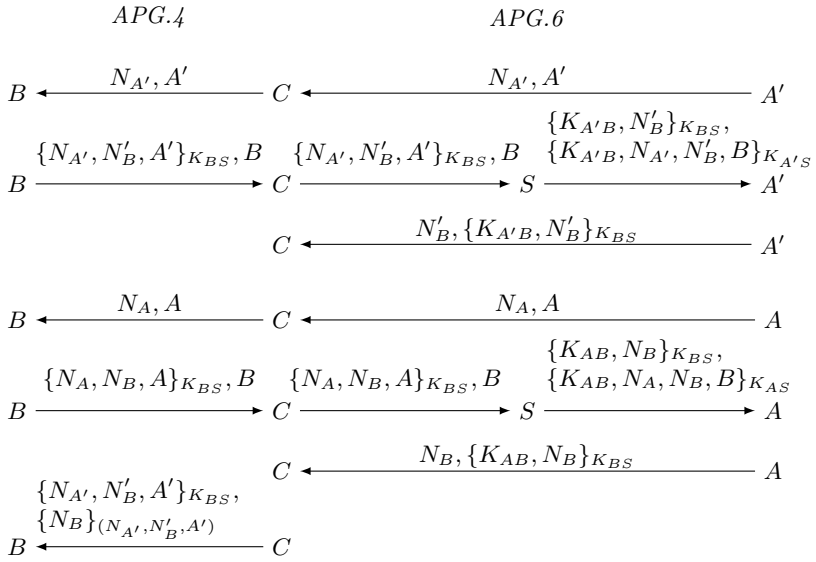

Fig. 5. Representation of the attack on APG.4 with APG.6.

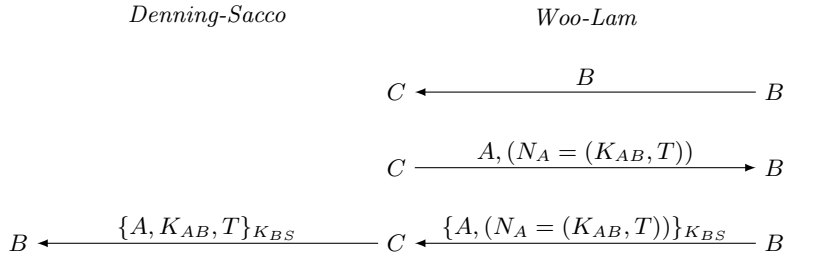

Fig. 6. Representation of the attack on Denning-Sacco with Woo-Lam.

APG.1 with APG.2: The attack described in [20] is an attack on the authentication of B, but we also found an attack on the authentication of A. In this attack, the intruder C plays the role of A in both protocols. First, B runs the *APG.2* [24] protocol as the initiator and then the protocol *APG.1* [24] as the responder. C can pretend to be A in *APG.1* and B will accept. In Figure 7 steps at the left are steps from *APG.1*, and the right part are steps from *APG.2*.

APG.3 with APG.2: This attack is an attack on the authentication of B. This attack is possible if A runs the *APG.3* [24] protocol as the initiator, and *APG.2* [24] as the responder. In this attack, C plays the roles of B and S in both protocols. Then C can pretend to be B in *APG.3*, and A will accept. In Figure 8 steps at the left are steps of *APG.3*, and at the right part are steps from *APG.2*.

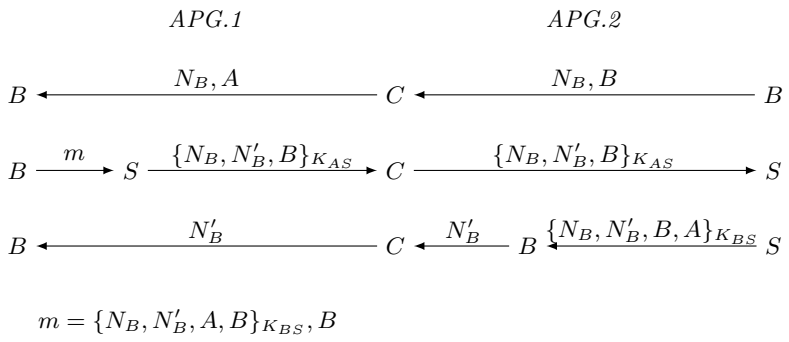

Fig. 7. Representation of the attack on APG.1 with APG.2.

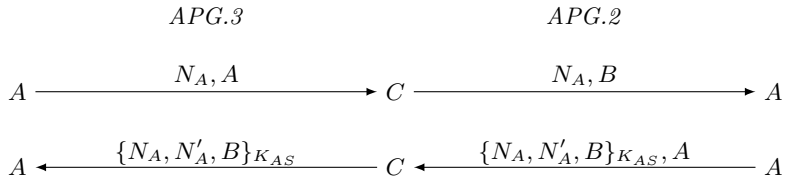

Fig. 8. Representation of the attack on APG.3 with APG.2.

We also found an attack on the authentication of A. In this attack, the intruder C plays the role of A in both protocols. B runs the *APG.2* protocol as the initiator, and *APG.3* as the responder. C can pretend to be A, and B in *APG.3* will accept. In Figure 9 steps at the left are steps from *APG.3*, and steps at the right are steps from *APG.2*.

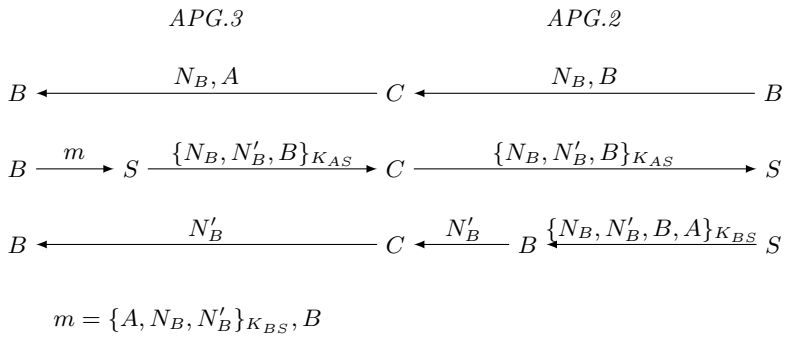

Fig. 9. Representation of the attack on APG.3 with APG.2.

APG.4 with APG.6: We found an attack on the authentication of B. In this attack, A initiates the protocol *APG.3*, then the intruder C will initiate *APG.6* with B, using data sent by A in the other protocol. Finally, C sends the answer of B to the server in *APG.4*, and lets the protocol run. In Figure 10, steps on the left hand side are steps from *APG.4*, and steps on the right hand side are steps from *APG.6*.

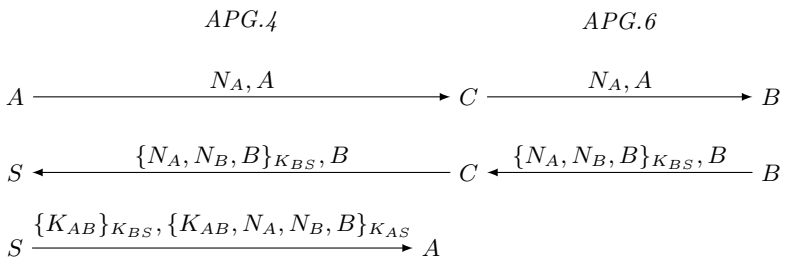

Fig. 10. Representation of the attack on APG.4 with APG.6.

This attack is possible because the message from *APG.6* used for this attack is also used in *APG.4*, so C can get a response from B, while B does not act in *APG.4*.

APG.5 with APG.6: This attack is on the authentication of A. In this attack, two protagonists A and A' initiate the *APG.6* [24] protocol with B, and the intruder C pretends to be A in *APG.5* [24]. In the protocol initiated by A', C learns $(N_{A'}, N'_B, A')$, used as a session key, and its encrypted version $\{N_{A'}, N'_B, A'\}_{K_{BS}}$. In the protocol initiated by A, C learns the nonce N_B, used to authenticate to B. In Figure 11, steps at the left part are steps of *APG.5*, and steps on the right are steps from *APG.6*.

This attack is a type-flaw attack. So we changed our model to disable such type-flaw attacks in Tamarin to see if there are other types of attacks, and we did not find another attack on the authentication of A.

We also found an attack on the authentication of A where the intruder uses $(N_{A'}, N'_B, A')$ as a session key. In this attack, A initiates the protocol *APG.5*, then the intruder C will initiate *APG.6* with B, using data sent by A in the other protocol. Finally, C sends the answer of B to the server in *APG.5*, and lets the protocol run. In Figure 12, steps on the left hand side are steps from *APG.5*, and on the right hand side are steps from *APG.6*.

This attack is possible because the message from *APG.6* used for this attack is also used in *APG.5*, so C can get a response from B, while B does not act in *APG.5*.

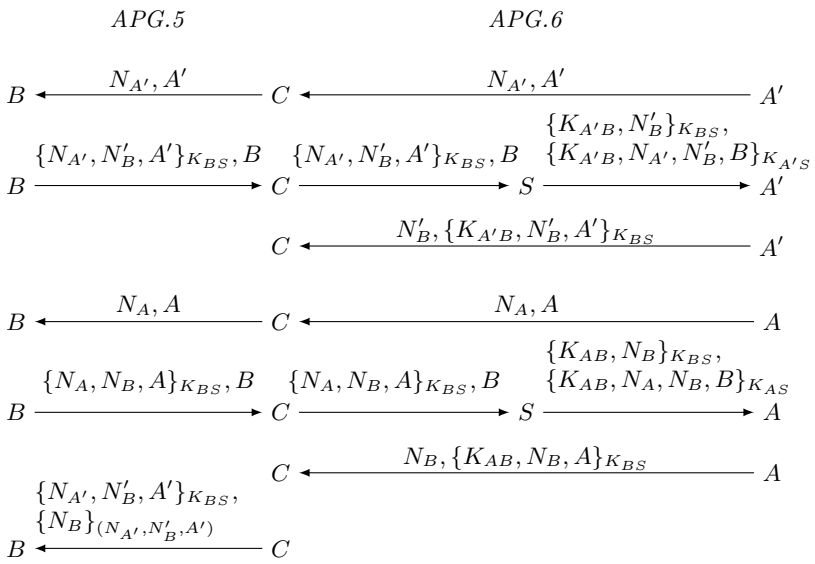

Fig. 11. Representation of the attack on APG.5 with APG.6.

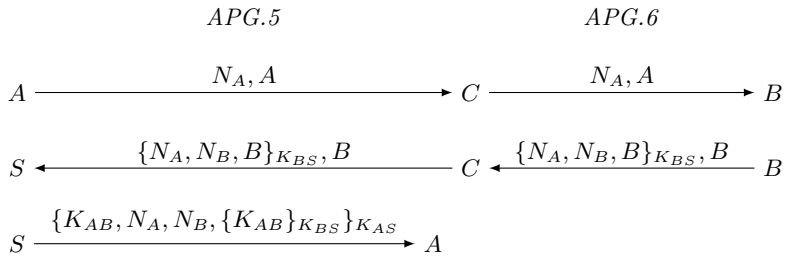

Fig. 12. Representation of the attack on APG.5 with APG.6.

3 Workflow in Tamarin

As we had to write many different combinations of multiple protocols to obtain our results, we tried to simplify the process by adopting the following workflow to combine to protocols:

1. Specify each protocol individually and check the properties in isolation using Tamarin.
2. Generate the files for all the required combinations using the individual specifications.
3. Verify the combined protocols, and compare the results to known results.

To simplify the process of generating the combined specifications, we adopted certain (mostly syntactic) conventions when specifying the protocols. These

mostly concern the common setup rules (key distribution etc.), the placement of labels, and uniqueness of labels to avoid conflicts.

These conventions allowed us to develop an algorithm that can generate the input files of the composed protocols based on the individual specifications, including intermediate lemmas that simplify the analysis for Tamarin. The generation of these lemmas goes beyond a pure syntactical merger of the individual files. The algorithm requires some interaction with Tamarin, but noticeably simplifies the following analysis. The main idea is that if Tamarin finds an a problem in the merged lemma, then we need to analyze the trace produced by the tool and to modify the lemma. This procedure seems to be systematic for all the examples that we have considered here.

This algorithm is not implemented yet, but we have tested it manually on combinations from [20], and also from [10]. In all these cases, we always succeed to make a valid intermediate lemma that removes all undesirable cases. We have also tested it on a tree-protocol-attack from [10], and succeed (by running the algorithm with the first and second protocol in a first step, and with this combination and the third in a second step). So this algorithm seems to work for multi-protocol-compositions with more than two protocols.

For more information about this algorithm, see Appendix A.

4 Conclusion

In this paper, we perform an automated analysis of multi-protocols in Tamarin. For this we have used the both protocols studied in [10] using Scyther and the protocol studied in [20] manually. We see that the tool finds different and new attacks in some cases. We also proposed an algorithm to systematically merge two Tamarin files for our analysis.

Our future work is to see how we can integrate our algorithm for automatically merging two Tamarin files into the tools in order to facilitate the life of Tamarin users. Finally our experience also shows us that it might even be possible to propose a similar heuristic to help Tamarin users by automatically generating such helping intermediate lemmas.

References

1. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Softw. Eng.*, 22(1):6–15, January 1996.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In *Proceedings of the 17th International Conference on Computer Aided Verification*, CAV'05, pages 281–285, Berlin, Heidelberg, 2005. Springer-Verlag.
3. Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA, 2001. IEEE Computer Society.

4. Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, February 1990.
5. Levente Buttyan, Sebastian Staamann, and Uwe Wilhelm. A simple logic for authentication protocol design. In *In 11th IEEE Computer Security Foundations Workshop*, pages 153–162. IEEE Computer Society Press, 1998.
6. John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0, 1997.
7. John A. Clark and Jeremy Jacob. On the security of recent protocols. *Inf. Process. Lett.*, 56(3):151–155, 1995.
8. Cas Cremers and Sjouke Mauw. *Security Properties*, pages 37–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
9. Cas J.F. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128, New York, NY, USA, 2008. ACM.
10. C.J.F. Cremers. Feasibility of multi-protocol attacks. In *Proc. of The First International Conference on Availability, Reliability and Security (ARES)*, pages 287–294, Vienna, Austria, April 2006. IEEE Computer Society.
11. C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, pages 139–161, 2006.
12. Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, August 1981.
13. Blot Elliott, Jannik Dreier, and Pascal Lafourcade. Formal Analysis of Combinations of Secure Protocols (Extended Version). Technical report, LORIA - Université de Lorraine, July 2017.
14. Tzonelih Hwang and Yung-Hsiang Chen. On the security of splice/as - the authentication system in wide internet. *Inf. Process. Lett.*, 53(2):97–101, January 1995.
15. I.-Lung Kao and Randy Chow. An efficient and secure authentication protocol using uncertified keys. *SIGOPS Oper. Syst. Rev.*, 29(3):14–21, July 1995.
16. John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 91–104, London, UK, UK, 1998. Springer-Verlag.
17. Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, November 1995.
18. Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44. IEEE Computer Society, 1997.
19. Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of computer security*, 7(2-3):89–146, 1999.
20. Anish Mathuria, Aditya Raj Singh, P. Venkat Shravan, and Rohit Kirtankar. Some new multi-protocol attacks. In *Proceedings of the 15th International Conference on Advanced Computing and Communications*, ADCOM '07, pages 465–471, Washington, DC, USA, 2007. IEEE Computer Society.
21. Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 696–701, 2013.
22. R M Needham and M D Schroeder. Authentication revisited. *SIGOPS Oper. Syst. Rev.*, 21(1):7–7, January 1987.

23. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.
24. A. Perrig and D. Song. Looking for diamonds in the desert - extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Workshop on Computer Security Foundations*, CSFW '00, pages 64–, Washington, DC, USA, 2000. IEEE Computer Society.
25. Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *J. Comput. Secur.*, 9(1-2):47–74, January 2001.
26. Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *SIGOPS Oper. Syst. Rev.*, 28(3):24–37, July 1994.
27. Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara. The design and implementation of an authentication system for the wide area distributed environment. *IEICE TRANSACTIONS on Information and Systems*, 74(11):3902–3909, 1991.
28. Hongbin Zhou and Simon N. Foley. Fast automatic synthesis of security protocols using backward search. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, FMSE '03, pages 1–10, New York, NY, USA, 2003. ACM.

A Details of the workflow

We denote by `In_dec#i` labels that manage the input in deconstructions lemma (each one can appear only once in rules), and `Out_dec#i` denotes labels that manage the output in deconstructions lemma (can appear many times in rules).

Making the combination of two protocols with Tamarin is not just to put all transitions from these protocols in the same file. But in fact we have often to solve some partial deconstruction problems. A partial deconstruction is when Tamarin cannot say where a fact come from. That makes open chains and Tamarin may not terminate. An idea is to make the conjunction of both deconstructions lemmas. With this, There are no more partial deconstruction left, but the lemma might be wrong, due to new sources in the other protocol, for rules concerned by the deconstructions lemma. So we have to correct the lemma and verify it again. The simplest way to do this is to add the counter-example given by Tamarin in the deconstruction lemma (by adding labels missing, to make the trace valid), but it is not sufficient for all cases, because we can create loop at the generation of the refined sources, and Tamarin does not terminate. This special case appears when we have in a rule labels `[In_dec1(n1), Out_dec2(n2)]` and in an other `[Out_dec1(n1), In_dec2(n2)]`. In this case a solution is to merge labels, that means that these two rules have labels `[In_dec12(n1, n2)]` (and no more `Out_dec1(n1)` or `Out_dec2(n2)`) and all labels `Out_dec2(n2)` and `Out_dec2(n2)` become `Out_dec12(n1, n2)` (include those in the deconstruction lemma). For example:

```
rule1:
[ In(<A, na, nb>) ]
--[ In_dec1(na, nb) ]->
```

```
 [ Out(m1) ]

 rule2:
 [ In(<A, na, nb>) ]
 --[ In_dec2(na, nb) ]->
 [ Out(m2) ]

 lemma deconstructions [sources]:
   "(All na nb #i. In_dec1(na, nb) @i ==>
    ((Ex #j. Out_dec1(na, nb) @j & #j < #i) |
     (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))
   &(All na nb #i. In_dec2(na, nb) @i ==>
    ((Ex #j. Out_dec2(na, nb) @j & #j < #i) |
     (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))"
```

Become:

```
 rule1:
 [ In(<A, na, nb> ]
 --[ In_dec(na, nb) ]->
 [ Out(m1) ]

 rule2:
 [ In(<A, na, nb> ]
 --[ In_dec(na, nb) ]->
 [ Out(m2) ]

  lemma deconstructions [sources]:
   "(All na nb #i. In_dec(na, nb) @i ==>
    ((Ex #j. Out_dec(na, nb) @j & #j < #i) |
     (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))"
```

It is also possible to create new loop by making the combination of both protocols, so Tamarin may not terminate the proof of lemmas that terminates when the protocol is alone.

- First we need to write "`theory [name] begin`"
- Copy builtins, functions and equations of both protocols without redundancies, we make the union of them.

```
def CopyBuiltins:
  put "builtins:" in [filename].spthy;
  for i in builtins
    put i in [filename].spthy;
    put "\n" in [filename].spthy;
  put "functions:" in [filename].spthy;
  for i in functions
    put i in [filename].spthy;
```

```
    put "\n" in [filename].spthy;
  put "equations:" in [filename].spthy;
  for i in equations
    put i in [filename].spthy;
    put "\n" in [filename].spthy;
```

– Copy the initialisations rules once.

```
def CopyInitialisation:
  for r in Rules from first protocol:
    if Is_Initialisation_Rules(r):
      put r in [filename].spthy;
```

– Copy rules of the first protocol, then copy rules of the second protocol without creating some conflicts with the names of rules, facts, and labels (keep the names of facts from initialisations rules).

```
def RenameFacts:
  for each Facts, Rules, Labels in Rules
                  from second protocols:
    if not in Initiation rules:
      rename them by Sec[actualName];
```

– Make the conjunction of both deconstruction lemmas.

```
def Deconstruction:
  put "lemma deconstruction[sources]:"
                  in [filename].spthy;
  copy lemma content from first protocol
                    in [filename].spthy;
  put "&" in [filename].spthy;
  copy lemma content from second protocol
                     in [filename].spthy;
```

– If we have two lines of the deconstructions lemma with the same arguments, and both rules including `In_dec#i` have exactly the same input, merge them:

```
def Merge:
  for i,j in rule
    if "In_dec1(arg1)" & "In(m)" in i
      if "In_dec2(arg2)" & "In(m)" in j
        replace "In_dec1(arg1)" by "In_dec(arg1,arg2)"
                                   in rules and lemmas;
        replace "In_dec2(arg2)" by "In_dec(arg1,arg2)"
                                   in rules and lemmas;
        replace "Out_dec1(arg1)" by "Out_dec(arg1,arg2)"
                                     in rules and lemmas;
        replace "Out_dec2(arg2)" by "Out_dec(arg1,arg2)"
                                     in rules and lemmas;
```

```
rule1:
[ In(<A, na, nb>) ]
--[ In_dec1(na, nb) ]->
[ Out(m1) ]

rule2:
[ In(<A, na, nb>) ]
--[ In_dec2(na, nb) ]->
[ Out(m2) ]

lemma deconstructions [sources]:
  "(All na nb #i. In_dec1(na, nb) @i ==>
   ((Ex #j. Out_dec1(na, nb) @j & #j < #i) |
    (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))
  &(All na nb #i. In_dec2(na, nb) @i ==>
   ((Ex #j. Out_dec2(na, nb) @j & #j < #i) |
    (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))"
```

Become:

```
rule1:
[ In(<A, na, nb> ]
--[ In_dec(na, nb) ]->
[ Out(m1) ]

rule2:
[ In(<A, na, nb> ]
--[ In_dec(na, nb) ]->
[ Out(m2) ]

 lemma deconstructions [sources]:
  "(All na nb #i. In_dec(na, nb) @i ==>
   ((Ex #j. Out_dec(na, nb) @j & #j < #i) |
    (Ex #j #k. #j < #i & #k < #i & KU(na) @j & KU(nb) @k)))"
```

– Copy restriction of protocols

```
def CopyRestriction:
  for r in restriction :
    put r in [filename].spthy;
```

– Try to prove the deconstructions lemma with Tamarin.

```
def Proof:
  run Tamarin [filename].spthy --prove=deconstruction;
```

We have two cases:

- The lemma is validate by Tamarin, it concludes the algorithm.
- Tamarin finds a counter-example, and gives us a trace where we have a `In_dec#i` without a `Out_dec#i` associated. So we add the missing

`Out_dec#i` in the rule preceding the rule with `In_dec#i` in the trace, then we retry to prove the deconstruction lemma.

```
def CorrectLemma:
  i = last rule with In_dec(x) in trace;
  j = previous rule of i in trace;
  put label Out_dec(x) in j;
```

Special case, if the arguments of `In_dec#i` are not in the the input of the rule, we put the `Out_dec#i` in the rule that sends the argument to the role.

```
def CorrectLemma:
  i = Last_Rule_With_In_dec(x) in trace;
  if x not in Out() of the rule
    z = last rule with Out(z1,x1,z2); //z1 and z2
    put label Out_dec(x) in z;        //can be null
  j = Previous_In_Trace(i);
  put label Out_dec(x) in j;
```

Other special case, if we have in a rule `[In_dec1(n1), Out_dec2(n2)]` and in an other `[Out_dec1(n1), In_dec2(n2)]`, we can merge dec1 and dec2, and change all `Out_dec1` and `Out_dec2` into `Out_dec12` in all other rules:

```
def Merge:
  for i,j in rule
    if "In_dec1(arg1)" & "Out_dec2(arg2)" in i
      if "In_dec2(arg2)" & "Out_dec1(arg1)" in j
        replace "In_dec1(arg1)" by "In_dec2(arg1,arg2)"
                                   in rules and lemmas;
        replace "In_dec2(arg2)" by "In_dec1(arg1,arg2)"
                                   in rules and lemmas;
        replace "Out_dec1(arg1)" by "Out_dec(arg1,arg2)"
                                    in rules and lemmas;
        replace "Out_dec2(arg2)" by "Out_dec(arg1,arg2)"
                                    in rules and lemmas;
```

This step has to be done before retry to prove the lemma, or Tamarin will loop at the generation of the refined sources.

– Copy lemmas you want to prove.
– Write "`end`"