J Autom Reasoning (2011) 46:353–388
DOI 10.1007/s10817-010-9185-y

Multi-Attacker Protocol Validation

Wihem Arsac · Giampaolo Bella · Xavier Chantry · Luca Compagna

Received: 16 June 2010 / Accepted: 17 June 2010 / Published online: 17 July 2010

Abstract Security protocols have been analysed focusing on a variety of properties to withstand the Dolev-Yao attacker. The Multi-Attacker treat model allows *each* protocol participant to behave maliciously intercepting and forging messages. Each principal may then behave as a Dolev-Yao attacker while neither colluding nor sharing knowledge with anyone else. This feature rules out the applicability of existing equivalence results in the Dolev-Yao model. The analysis of security protocols under the Multi-Attacker threat model brings forward yet more insights, such as *retaliation attacks* and *anticipation attacks*, which formalise currently realistic scenarios of principals competing each other for personal profit. They are variously demonstrated on a classical protocol, Needham-Schroeder's, and on a modern deployed protocol, Google's SAML-based single sign-on protocol. The general threat model for security protocols based on set-rewriting that was adopted in AVISPA (Armando et al. 2005) is extended to formalise the Multi-Attacker. The state-of-the-art model checker SATMC (Armando and Compagna, Int J Inf Secur 6(1):3–32, 2007) is then used to automatically validate the protocols under the new threats, so that retaliation and anticipation attacks can automatically be found. The tool support scales up to the

This work was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (www.avantssar.eu).

W. Arsac · X. Chantry · L. Compagna
SAP Research, Sophia Antipolis, France

G. Bella (✉)
Dipartimento di Matematica e Informatica,
Università di Catania, Catania, Italy
e-mail: giamp@dmi.unict.it

G. Bella
Software Technology Research Laboratory,
De Montfort University, Leicester, UK

Multi-Attacker threat model at a reasonable price both in terms of human interaction effort and of computational time.

Keywords Security protocols · Attacker models · Automated reasoning

1 Introduction

The analysis of security protocols stands among the most attractive niches of research in computer science, as it has attracted efforts from many communities. It is difficult to even provide a satisfactory list of citations, which would have to at least include process calculi [1, 38], strand spaces [23, 26], the inductive method [14, 36] and advanced model checking techniques [5, 20]. Any meaningful statement about the security of a system requires a clear specification of the threats that the system is supposed to withstand. Such a specification is usually referred to as *threat model*. Statements that hold under a threat model may no longer hold under other models. For example, Needham remarked that if the threat model only accounts for attackers that are outsiders, then Lowe's famous attack on the Needham-Schroeder Public-Key (NSPK) protocol cannot succeed [33], and the protocol may be claimed secure. But the protocol is notoriously insecure under a model that allows the attacker as a registered principal.

The standard threat model for symbolic protocol analysis is the Dolev-Yao model (DY in brief), which sees a powerful attacker control the whole network traffic. The usual justification is that a protocol secure under DY certainly is secure under a less powerful, perhaps more realistic attacker. By contrast, a large group of researchers consider DY insufficient because a DY attacker cannot do cryptanalysis, and their probabilistic reasoning initiated with a foundational paper [19]. This sparked off a research thread that has somewhat evolved in parallel with the DY research line, although some efforts exist in the attempt to conjugate them [2, 13, 27]. The present paper is not concerned with probabilistic protocol analysis. Our main argument is that security protocols may still hide important subtleties even after they are proved correct under DY. These subtleties can be discovered by symbolic protocol analysis under a new threat model that perhaps is more adherent to the present real world than DY is.

The new model we develop here is the *Multi-Attacker* (MA in brief), which features each protocol participant as a DY attacker who does not collude or share knowledge with anyone else. In MA, it is meaningful to continue the analysis of a protocol *after* an attack is mounted, or to anticipate the analysis by looking for extra flaws *before* an initiated attack terminates, something that has never been seen in the relevant literature. This can assess whether additional attacks can be mounted either by the same principal who attacked in the first place or by different attackers. Even novel scenarios whereby principals attack each other are supported. A significant scenario is that of *retaliation* [18], where an attack is followed by a counterattack. This paper recasts that scenario into the MA threat model, and expands it. Also, a completely new scenario is defined, that of *anticipation*, where a principal's attack is anticipated, before its termination, by another attack by some other principal.

The new threat model is used to analyse a classical protocol, Needham-Schroeder's public-key protocol, and a modern deployed one, Google's single sign-on protocol. Various findings are described about the former, including an exhaustive

analysis of the malicious activity that may (successfully) follow Lowe's man-in-the-middle attack. An attack has recently been reported on Google's single sign-on protocol: a malicious service provider may access a client's resources at Google [7]. Here, it is found that the attack can be retaliated: Google may then access the client's resources at the malicious service provider who attacked first. It becomes clear that our focus is on protocol niceties that go beyond standard confidentiality or authentication properties.

The formal language to support our analysis is found in an existing set-rewriting formalism suited for model checking, which in turn offers the necessary mechanical support. The language is IF, which can specify inputs to the AVISPA backends [5] and more specifically to SATMC, a successful SAT-based model checker [9]. After we specified the new threat model and the new properties of interest, we were prepared to face major performance issues. However, although the efficiency of SATMC was put at stake by the new properties to validate, the runtimes never exceeded few minutes. Remarkably, while the tool always validated our pre-existing informal analysis, it also provided various unknown insights. It then seems fair to state that the tool supports well the validation of security protocols under modern everyone-for-themselves threats.

This manuscript conjugates and significantly expands on two recent conference papers [10, 11]. It is structured as follows. The MA threat model is introduced (Section 2) and then used to informally analyse the two protocols (Section 3). The extensions that the new threat model require of the validation method are then presented (Section 4). After that, the actual validation experiments are described (Section 5) and the tool performance is outlined (Section 6). Some conclusions terminate the paper (Section 7).

2 MA: The Multi-Attacker Threat Model

DY can be considered the standard threat model to study security protocols [25]. The DY attacker controls the entire network although he cannot perform cryptanalysis. Some historical context is useful here. The model was defined in the late 1970s when remote computer communications were still confined to military/espionage contexts. It was then natural to imagine that the entire world would collude joining their forces against a security protocol session between two secret agents of the same country. The DY model has remarkably favoured the discovery of significant protocol flaws, but the prototype attacker has significantly changed today. To become an attacker has never been so easy as in the present technological setting because hardware is inexpensive, while security skill is at hand—malicious exploits are even freely downloadable from the web.

A seminal threat model called BUG [18], which dates back to 2002 [17], must be recalled here. The name is a permuted acronym for the "Good", the "Bad" and the "Ugly". This model attempts stricter adherence than DY's to the changed reality by partitioning the participants in a security protocol into three groups. The Good principals would follow the protocol, the Bad would in addition try to subvert it, and the Ugly would be ready to either behaviour. A simple byproduct of this taxonomy is the chance that a principal may change role and decide to attempt illegal exploitation of a protocol although he has always conformed to it so far. This

seems the first account in the literature of formal protocol verification on the chance that attackers may attempt to attack each other without sharing knowledge. More recently, Bella observed [15] that the partition of the principals had to be dynamically updated very often, in principle at each event of a principal's sending or receiving a message, depending on whether the principal respected the protocol or not. Thus, BUG appeared overly detailed, and he simplified it as the Rational Attacker threat model (RA in brief), which lets any principal make cost/benefit decisions at any time to either behave according to the protocol or not [15]. After BUG's inception [17], homologous forms of rational attackers were specifically carved out in the area of game theory [3, 21] and therefore are, as such, not directly related to our symbolic analysis.

Analysing a protocol under the Rational Attacker requires specifying each principal's cost and benefit functions, but this still seems out of reach for the current mechanised approaches, especially for classical model checking. By abstracting away the actual cost/benefit analysis, we recently derived a simplified model, the General Attacker (GA in brief), where any principal may behave as a Dolev-Yao attacker [11]. The change of perspective in RA or in GA with respect to DY is clear: principals do not collude for a common aim but, rather, each of them acts for his own personal sake. By contrast, a pair of colluding DY attackers is equivalent to a single DY attacker in terms of generated attacks, as confirmed by a formal proof [22]. Although there is no notion of collusion constraining these threat models, the human protocol analysers might define specific ones for their particular investigations. Endowing each principal with the entire potential of a DY attacker signifies that he may send any of the messages he can form to anyone. Such messages include both the legal ones, conforming to the protocol in use, and the illegal, forged ones, which he can build from the analysis of the traffic though without cryptanalysis.

It was recently suggested at an international venue [10] that GA is similar to DY provided that all principals reveal their secrets to the attacker. Such a threat model, which we address as DY+ appears equivalent to GA in terms of attacks that can be found: any illegal message that a principal may send in GA may be sent by the single DY+ attacker (exactly because he knows everyone's secrets). However, the two models do not seem trace-equivalent, as DY+ entangles the interpretation of attacks where principals attack each other. In fact, the single attacker will always be the originator of any attack, complicating the identification of the real perpetrator; for attacks against the attacker, the model will feature the attacker attacking himself, thus stretching the interpretation of the victim to an extreme. By contrast, as the sequel of this manuscript demonstrates, perpetrator and victim are naturally expressed in GA because its gist is exactly to reflect modern everyone-for-themselves scenarios.

The variety of experiments we have conducted thus far in validating security protocols under GA have produced a number of insights but also the trivial impersonation attacks due to a principal's deliberately revealing his long-term keys to someone else. This motivates an additional refinement of the model.

> The Multi-Attacker (MA) threat model: *each principal may behave as a Dolev-Yao attacker but will never reveal his long-term secrets.*

MA can be seen as a refinement of GA with some rationality that avoids the trivial impersonation attacks. This does not seem restrictive in the present epoch,

in the sense that it will not rule out significant attack scenarios, as well as DY did not seem restrictive in the early 1980s. Moreover, MA relates to DY+ as GA does. The findings obtained under GA [10, 11] of course continue to hold under MA.

As we shall see, analysing protocols under the MA threat model yields unknown scenarios of retaliation or anticipation. This paves the way for a future, more detailed analysis under RA. For example, if an attack can be retaliated under MA, such a scenario will not occur under RA because the cost of attacking clearly overdoes its benefit, and hence the attacker will not attack in the first place. This argument is after all not striking: even a proper evaluation of the "realism" of classical attacks found under DY would have required a proper cost/benefit analysis. A realistic consequence is that even a deployed protocol suffering an attack that can be retaliated may perhaps be kept in place.

3 Analysing Protocols under MA

We begin with an informal analysis of a classical protocol (Section 3.1) and of a modern deployed protocol (Section 3.2) under the new threat model.

3.1 A Classical Protocol

The BUG threat model was demonstrated over the public-key Needham-Schroeder protocol yielding an *indirect* retaliation attack [18]. In the sequel of this section, we recast that analysis under the MA model and extend it with other attacks including *direct* retaliation—these concepts are used on the basis of their intuitive meaning here, but will be formalised later (Section 4.3). The protocol version studied here, which we address as NSPK++, is its original design terminated with the completion steps for authenticated money transfers (Fig. 1). Clearly, all symbols in italic are variable names, as appropriate for a generic description of the protocol. The gist is that, once the peers authenticate each other, each of them may ask to debit his account, held at the other one, with someone else as a beneficiary. It can be seen that principal A issues a fresh nonce Na in message 1, which she sees back in message 2. Hence, A learns that B acted at the other end of the network because message 1 is encrypted under B's public key, the nonce was fresh and cryptanalysis cannot be broken by assumption. Messages 2 and 3 give B the analogous information about A by means of nonce Nb. Message 4a is a request from A at (bank) B to transfer $X1$€ from A's account $Y1$'s. Message 4b is the analogous request from B at A towards $Y2$.

$$
\begin{aligned}
1.\quad & A \rightarrow B : \{Na, A\}_{Kb} \\
2.\quad & B \rightarrow A : \{Na, Nb\}_{Ka} \\
3.\quad & A \rightarrow B : \{Nb\}_{Kb} \\
4a.\quad & A \rightarrow B : \{\text{“Transfer X1€ from A's account to } Y1\text{'s”}\}_{\langle Na,Nb\rangle} \\
4b.\quad & B \rightarrow A : \{\text{“Transfer X2€ from B's account to } Y2\text{'s”}\}_{\langle Na,Nb\rangle}
\end{aligned}
$$

Fig. 1 NSPK++: the NSPK protocol terminated with the completion steps

Incidentally, the figure also shows a notational convention used throughout this paper: angle brackets are used to denote messages whose outermost operator is concatenation. This protocol version is subject to Lowe's attack [30], as described in Fig. 2. It can be seen how the attacker C masquerades as A with B to carry out an illegal money transfer at B (which intuitively is a bank) from A's to C's account.

It is known that the problems originated with the confidentiality attack upon the nonce Nb. Another observation is that there is a second nonce whose confidentiality is violated—by B, not by C—in this scenario: it is Na. Although it is invented by A to be only shared with C, also B learns it. This does not seem to be an issue in the DY model, where all principals except C followed the protocol like soldiers follow orders. What one of them could do with a piece of information not meant for him therefore became uninteresting. To what extent this is appropriate to the current real world, where there often are various attackers with targets of their own, is at least questionable. Strictly speaking, B's learning of Na is a new attack because it violates the confidentiality policy upon the nonces, which are later used to form a session key. It can be easily captured in the MA threat model, where more than one principal may act illegally at the same time for their own sake.

We are facing a new perspective of analysis. Principal B did not have to act to learn a nonce not meant for him, therefore this is named an *indeliberate attack* [17]. To use a metaphor, B does not know which lock the key Na can open. This is not an issue in the MA threat model, where each principal just sends out anything he can send to anyone. Nevertheless, there are at least four methods, which would be worth of a cost/benefit analysis under the Rational Attacker, to help B practically evaluate the potential of Na.

1. There often is some *proximity relation* [16] between the principals executing a protocol, who can be seen as a community that interacts by means of the protocol. A community might be a department, a company, a group of companies, etc. So B has a useful indication that Na can be exploited exactly within the community, not just everywhere on the Internet, which would be impractical. To continue with the metaphor, he might then try the key Na at all locks on his floor.
2. Because B sees the entire network traffic, in particular he sees the session between A and C and therefore realistically makes the right guess about nonce Na.

$$
\begin{array}{lll}
1. & A \to C : \{Na, A\}_{Kc} & \\
 & & 1'. \quad C(A) \to B : \{Na, A\}_{Kb} \\
 & & 2'. \quad B \to A : \{Na, Nb\}_{Ka} \\
2. & C \to A : \{Na, Nb\}_{Ka} & \\
3. & A \to C : \{Nb\}_{Kc} & \\
 & & 3'. \quad C(A) \to B : \{Nb\}_{Kb} \\
 & & \\
4a'. & C(A) \to B : \{\text{“Transfer 1000€ from } A\text{'s account to } C\text{'s”}\}_{\langle Na,Nb\rangle} &
\end{array}
$$

Fig. 2 Lowe's attack to the NSPK++ protocol

3. Principal B may use an *out-of-band detection challenge* [18] to learn whether something went wrong with the session upon Na and Nb, which he believes to have executed with A. Thus, B can attempt the following dull transfer at A for all principals X:

$$B(X) \rightarrow A : \{\text{“Transfer 1€ from } X\text{'s account to } B\text{'s”}\}_{\langle Na,Nb\rangle}$$

   As one transfer succeeds—and B can realise this—he detects the attacker C who mounted Lowe's attack.
4. A simple alternative is that B merely waits for A's complaint that she has noticed a money transfer towards C, which she did not require herself.

The natural consequence of B's learning Na is the *retaliation attack* [18] in Fig. 3. Note that the first method indicated above gives B a reasonable set of target principals to try retaliation against, while the second one gives him a probabilistic answer originated from traffic analysis. However, the remaining two methods exactly tell him who the target for retaliation is.

It is worth remarking once more that a retaliation attack cannot be captured in the standard DY threat model, where all potential attackers merely collude to form a super-potent one, but the MA model can support this notion.

An anonymous reviewer observed that the notion of retaliation scenario might be related to the notion of *balanced* protocol [29]. Evaluating this formally is beyond the focus of our paper, but it can be appreciated that a retaliation scenario is more general as it can be instantiated to any attack. This will become clearer below (Section 4.3).

Having glimpsed at how B can exploit his knowledge of Na, we may also wonder whether principal C may at all become a victim. These questions are relevant today, when hackers routinely attack each other, and can be formally answered in the MA threat model. We begin informally but systematically by analysing the six permutations of the three principals involved. They will reveal who can attack whom using an appropriate instance of the protocol messages 4a or 4b.

The first move is to state the beliefs of principals upon the pair of nonces (that form a session key) at the end of Lowe's scenario, that is after the trace $1, 1', 2', 2, 3, 3'$.

Belief i B believes that the nonce pair is shared with A and not with C. Indeed, B replied to a session apparently with A and completed the protocol successfully.

Belief ii A believes that the nonce pair is shared with C and not with B. Indeed, A chose to initiate a session with C and noticed no irregularity.

Belief iii C believes that the nonce pair is shared with A *and* with B. Indeed, C knows the threat model—see below.

$$4b.\quad B(C) \rightarrow A : \{\text{“Transfer 2000€ from } C\text{'s account to } B\text{'s”}\}_{\langle Na,Nb\rangle}$$

Fig. 3 Retaliation attack following Lowe's attack

The first two beliefs are rather obvious. The third means that C assumes the worst case that everyone will attempt exploitation of his knowledge for personal profit. In particular C himself handed the nonce pair over to B, and therefore knows that B may attempt to exploit such a knowledge in every possible way.

We are now ready for the second move, a systematic analysis. Here are the six possible attack attempts in terms of an illegal money transfer, which may follow Lowe's scenario. They are obtained by permuting the three involved principals.

Attempt I $C(A) \rightarrow B : \{\text{Dear } B, \text{move 1,000€ from } A\text{'s account to } C\text{'s}\}_{\langle Na,Nb\rangle}$

Attempt II $B(C) \rightarrow A : \{\text{Dear } A, \text{move 1,000€ from } C\text{'s account to } B\text{'s}\}_{\langle Na,Nb\rangle}$

Attempt III $A(C) \rightarrow B : \{\text{Dear } B, \text{move 1,000€ from } C\text{'s account to } A\text{'s}\}_{\langle Na,Nb\rangle}$

Attempt IV $B(A) \rightarrow C : \{\text{Dear } C, \text{move 1,000€ from } A\text{'s account to } B\text{'s}\}_{\langle Na,Nb\rangle}$

Attempt V $C(B) \rightarrow A : \{\text{Dear } A, \text{move 1,000€ from } B\text{'s account to } C\text{'s}\}_{\langle Na,Nb\rangle}$

Attempt VI $A(B) \rightarrow C : \{\text{Dear } C, \text{move 1,000€ from } B\text{'s account to } A\text{'s}\}_{\langle Na,Nb\rangle}$

The concluding move is a number of propositions that establish whether the mentioned attempts succeed as attacks or not. Because C is the only principal who acted illegally until the end of Lowe's scenario, our analysis begins with C's attack and the corresponding retaliation. It is useful to recall that given an attack, we can define its direct retaliation attack, as carried out by the former victim against the former attacker, and its indirect retaliation attack, as carried out by some former third party against the former attacker [18].

Proposition 1 *Attempt* I *succeeds as an attack—it is Lowe's attack* [30]. *Proof by Belief* i.

Proposition 2 *Attempt* II *succeeds as an attack—it is a published indirect retaliation attack of Lowe's attack* [18]. *Proof by Belief* ii.

Proposition 3 *Attempt* III *does not succeed as an attack—it remains an attempt at direct retaliation of Lowe's attack. Proof by Belief* i.

Proposition 4 *Attempt* IV *succeeds as an attack—it is an unpublished attack that we name "our attack". Proof by Belief* iii*: C cannot discern the real sender.*

Proposition 5 *Attempt* V *does not succeed as an attack—it remains an attempt at indirect retaliation of our attack. Proof by Belief* ii.

Proposition 6 *Attempt* VI *succeeds as an attack—it is an unpublished direct retaliation attack of our attack. Proof by Belief* iii*: C cannot discern the real sender.*

Having analysed exhaustively all possible attacks following Lowe's man-in-the-middle scenario, we were confident not to be missing any extra insights in this scenario, not even under the MA threat model. However, the mechanical validation that followed taught us that we were wrong (Section 5).

Moreover, an anonymous reviewer correctly noted that the attack attempts make sense not only for the permutations of the principals but also for other combinations where the real sender and the receiver are the same. For example,

$$A(C) \rightarrow A : \{\text{Dear } A, \text{ move 1,000€ from } C\text{'s account to } Y\text{'s}\}_{\langle Na,Nb\rangle}$$

trivially succeeds, and lets A steal money from C's account for the benefit of any principal Y's. For example, A might want to build this message by herself for audit purposes and feel safe in instructing an illegal money transfer. Of course, it might even be the case that $Y = A$. This kind of attack is somewhat simpler to construct than the previous ones because it only relies on a pair of peers and may therefore follow a single protocol session, not necessarily Lowe's scenario. Not surprisingly, we found out that it can also be validated more quickly, and hence our validation section (Section 5) omits it for brevity.

3.2 A Modern Deployed Protocol

The work discussed in [7] provides us with a modern and industrially-relevant security protocol on which to experiment with the MA threat model. The SAML-based Single Sign-On (SSO) protocol for Google Apps offers Google's customer enterprises a direct and transparent access to popular web-based applications like Gmail or Google Calendar. SAML 2.0 [35] from OASIS is the de-facto standard in the context of SSO. This is why Google and many other major software providers are basing their SSO solutions on SAML. The first version of the SAML-based Single Sign-On (SSO) protocol for Google Apps released by Google in February 2007 is in Fig. 4, which needs some explanation. In the following, we will refer to this protocol version as *Google SSO*. The arrows should be interpreted as follows:

- $\circ\!\!\rightarrow\!\!\bullet$ The channel is confidential for the intended receiver (i.e., its output is exclusively accessible to the receiver) and weakly authentic for the sender (i.e., its input is exclusively accessible to a single, yet unknown, sender);
- $\bullet\!\!\rightarrow\!\!\circ$ The channel is weakly confidential for the intended receiver (i.e., its output is exclusively accessible to a single, yet unknown, receiver) and authentic for the sender (i.e., its input is exclusively accessible to the sender);
- $\bullet\!\!\rightarrow\!\!\bullet$ The channel is confidential for the intended receiver and authentic for the sender.

$$
\begin{array}{lllll}
1. & C & \circ\!\!\rightarrow\!\!\bullet & SP & : C, SP, URI \\
2. & SP & \bullet\!\!\rightarrow\!\!\circ & C & : C, IdP, \text{AuthReq}(ID, SP), URI \\
3. & C & \circ\!\!\rightarrow\!\!\bullet & IdP & : C, IdP, \text{AuthReq}(ID, SP), URI \\
4. & IdP & \bullet\!\!\rightarrow\!\!\bullet & C & : SP, \{|\text{AuthAssert}(C, IdP)|\}_{\mathrm{K}^{-1}_{IdP}}, URI \\
5. & C & \circ\!\!\rightarrow\!\!\bullet & SP & : SP, \{|\text{AuthAssert}(C, IdP)|\}_{\mathrm{K}^{-1}_{IdP}}, URI \\
6. & SP & \bullet\!\!\rightarrow\!\!\circ & C & : Resource
\end{array}
$$

Fig. 4 SAML-based SSO protocol for Google Apps

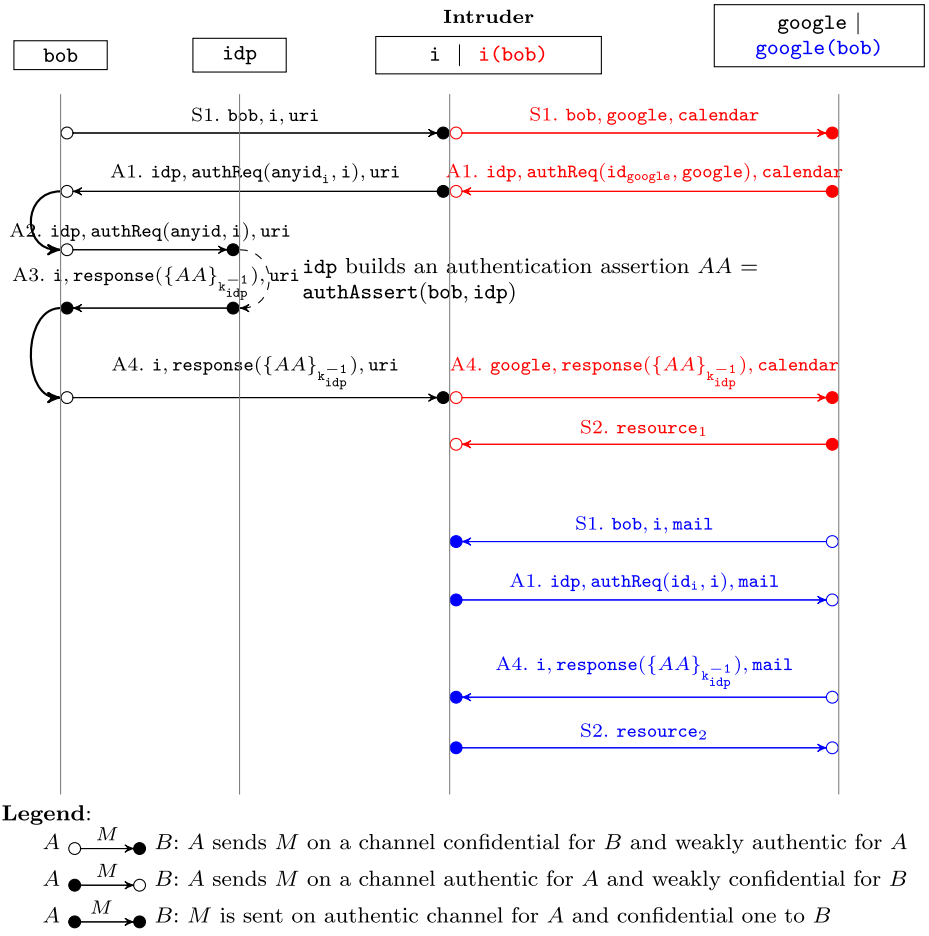
Three roles are defined: a client C, an identity provider IdP and a service provider SP. The client C, typically a web browser, wishes to get access to a service or a resource provided by SP. Then, IdP authenticates C and issues a corresponding authentication assertion that is consumed by SP to give C access to the requested resource, which is transmitted confidentially at the end of the protocol.

More in detail, the protocol executes as follows. First, C requests SP to provide the resource located by its URI address. SP replies with an authentication request AuthReq(ID, SP)—where ID uniquely identifies the request—that is redirected by C to IdP. Then IdP defines an authentication assertion $AA = \text{AuthAssert}(C, IdP)$ based on C valid credentials and builds a response message $SP, \{|AA|\}_{\mathrm{K}_{IdP}^{-1}}, URI$, where $\{|AA|\}_{\mathrm{K}_{IdP}^{-1}}$ is the assertion digitally signed with K_{IdP}^{-1}, the private key of IdP. The response message reaches SP via C, and finally SP can make available at the fresh address *Resource* what C originally requested.

It is important to notice that the message exchanged through the protocol presented above are protected at the transport layer. Inspired by [32], arrows augmented with empty or filled-in circles indicate the assumptions on the communication channel. Of course, these assumptions must be carefully considered for any realistic analysis of the protocol. For example, they have been embedded in a recent validation work [7]. The same work reported a serious security flaw in Google SSO [7]. The attack, meanwhile fixed, allowed a (dishonest) service provider to access resources of Google Apps under the identity of an unaware user. Two critical security properties the protocol was expected to enforce are thus violated: (1) SP authenticates C, that is SP has to be sure it has been talking with C; and (2) *Resource* must be kept secret between C and SP. The problem was due to the absence of a few but critical fields in the authentication assertion generated by the IdP.

In experimenting with the MA threat model, we found out that this attack can be retaliated. Figure 5 shows both the original attack and the four additional messages that allow the service provider Google, this time, to retaliate by accessing resources of the initially dishonest service provider. Let us discuss the entire trace in detail. In the original attack, `bob` initiates a session of the protocol to access a resource provided by the service provider `i`. In parallel, `i` starts a new session of the protocol with the service provider `google` pretending to be `bob` (indicated as `i(bob)`) and mischievously reuses the authentication assertion received by `bob` (cf. step A4) to trick `google` into believing he is `bob`. The attack terminates with the delivery of the resource (whose access should be reserved to `bob`) to `i`. In the retaliation attack, `google` starts another session as a client with the service provider `i`, pretending to be `bob`. By reusing the authentication assertion received by `i(bob)`, `google` is able to access a resource reserved to `bob` provided by `i`.

A possible business case for this retaliation is discussed hereafter. Let `idp` be an enterprise that is appointed to thoroughly evaluate and compare Google Apps (provided by `google`) against its competitor market products, including Zimbra for example (provided by `i`). A group of experts working at `idp`, and among them `bob`, will deeply inspect one by one these products under the premises that no result about product P will be disclosed outside the perimeter of P's provider without P's provider authorization. In this context, a SSO solution with `idp` serving as identity provider significantly simplifies things. Operationally, Google SSO is the deployed protocol. Clearly, any service provider, once obtained by `idp` an authentication assertion for `bob`, would be able to access any other service provider under the



Legend:

$A \xrightarrow{M} B$: A sends M on a channel confidential for B and weakly authentic for A

$A \xrightarrow{M} B$: A sends M on a channel authentic for A and weakly confidential for B

$A \xrightarrow{M} B$: M is sent on authentic channel for A and confidential one to B

Fig. 5 Original attack on Google SSO and retaliation

identity of bob to have a look at the recent discoveries bob made on competitor products.

This scenario can be interpreted in a similar fashion as Lowe's interpretation of his attack in a banking environment. However, here the victim is the client bob in both cases, once at each service provider. So, this scenario seems some sort of *conjuring* retaliation because a victim, once attacked, can be attacked again by other principals exploiting the first attack. This concept is formalised below (Section 4.3).

4 Extending the Validation Method over MA

We have used one of the AVISPA backends to perform our experiments under MA: the SAT-based model checker SATMC [8]. Its core is a procedure that automatically generates propositional formulae whose satisfying assignments (if any) correspond

to counterexamples (i.e. execution traces of M that falsify G) of length bounded by some integer k, which can be iteratively deepened. The length k represents the maximum number of transitions, which includes both the standard protocol steps and the actions of the attacker.[1] Finding violations (of length k) on protocol properties boils down to solving propositional satisfiability problems. SATMC accomplishes this task by invoking state-of-the-art SAT solvers, which can handle in most of the cases satisfiability problems with hundreds of thousands of variables and clauses.

SATMC has successfully tackled the problem of determining whether the concurrent execution of a finite number of sessions of a protocol enjoys certain security properties in spite of the DY attacker [6, 9]. It is useful to outline that work here. The security protocol general verification problem is well known to be undecidable. Still, for many protocols, the verification effort can be limited to a bounded number of sessions [31]. Moreover, even for those protocols that should be checked under an unbounded number of concurrent protocol executions, violations in their security requirements often exploit only a small number of sessions. In addition, when the number of sessions is bounded, cycles (if any) in the execution of honest agents may be replaced by a fixed number of steps representing some iterations of the loop. For these reasons, it is often sufficient to consider a finite number of sessions in which each agent performs a fixed number of steps. In such situations, the correspondent protocol insecurity problem belongs to the NP-complete complexity class [37]. SATMC guarantees termination on this class of protocols while it can be used as a semi-decision procedure for all other protocols.

Leveraging on that work, we aim at relaxing the assumption of a single, superpotent attacker to specify the MA threat model, where principals can even compete each other. As demonstrated above, our aim is to study novel protocol subtleties that go beyond the violation of standard security properties such as confidentiality and authentication. Therefore, we shall recast those notions as a model checking problem in the following.

4.1 Basics of SAT-Based Model Checking

This subsection outlines the basic definitions and concepts underlying SAT-based model checking. The reader who is familiar with such concepts can skip this. Let us recall that a model checking problem can be stated as

$$M \models (C_I \Rightarrow G) \tag{1}$$

where M is a labelled transition system modelling the initial state of the system and the behaviours of the principals (including their malicious activity), C_I is a conjunction of LTL formulae capturing assumptions on communication channels, and G is an LTL formula expressing the security property to be checked.

The states of M are represented as sets of *facts* i.e. ground atomic formulas of a first-order language with sorts. If S is a state, then its facts are interpreted as the propositions holding in the state represented by S, all other facts being false in that state (closed-world assumption). A state is written down by the convenient syntax of

[1] Notice that the user may specify an infinite upper bound. In that case the procedure is not guaranteed to terminate.

a set of facts separated by the . symbol, as we shall see. The . symbol is used by us as set constructor.

The transitions of M are represented as *set-rewriting rules* of the form $(P \,.\, N \xrightarrow{rl(v_1,\ldots,v_n)} R)$, where P and R are finite set of facts, N is a finite set of negated facts $\neg f_1 \,.\, \ldots \,.\, \neg f_n$, rl is a *rule label*, i.e. a function symbol uniquely associated with the rule, and $v_1, \ldots, v_n$ are the variables occurring in P. It is required that the variables occurring in N and in R also occur in P. The rule label expresses what the rule is there for: for example, the label $\texttt{step}_i$ is for a rule that formalises the i-th legal protocol step, the label `overhear` is for a rule whereby an attacker reads some traffic, and so on. We will encounter a number of self-explaining rule labels in the sequel.

Let S be a set of facts, $(P \,.\, N \xrightarrow{rl(v_1,\ldots,v_n)} R)$ a rewrite rule and σ a substitution of the variables $v_1, \ldots, v_n$. We say that *rule (instance)* $\rho = rl(v_1\sigma, \ldots, v_n\sigma)$ *is applicable in state* S if and only if $P\sigma \subseteq S$ and $f\sigma \notin S$ for each f such that $\neg f \in N$. Intuitively a rewrite rule can fire in a state anytime its positive facts hold in the state while its negative ones do not. If $\rho = rl(v_1\sigma, \ldots, v_n\sigma)$ is applicable in S, then $S' = \mathrm{app}_\rho(S) = (S \setminus P\sigma) \cup R\sigma$ is the state resulting from the execution of ρ in S. A *path* π is an alternating sequence of states and rules $S_0\rho_1S_1 \ldots S_{n-1}\rho_nS_n$ such that $S_i = \mathrm{app}_{\rho_i}(S_{i-1})$ (i.e. S_i is a state resulting from the execution of ρ_i in S_{i-1}), for $i = 1, \ldots, n$. Let $\mathcal{I}$ be the initial state of the transition system; if $S_0 \subseteq \mathcal{I}$, then we say that the path is *initialised*. Let $\pi = S_0\rho_1S_1 \ldots S_{n-1}\rho_nS_n$ be a path. We define $\pi(i) = S_i$ and $(\pi, i) = S_i\rho_{i+1}S_{i+1} \ldots S_{n-1}\rho_nS_n$. Therefore, $\pi(i)$ and (π, i) are the i-th state of the path and the suffix of the path starting with the i-th state respectively. Also, it is assumed that paths have infinite length even if they are defined as a finite alternating sequence of states and rules. This is obtained by assuming stuttering transitions in the transition system and it is useful to apply standard LTL semantics in our setting.

The language of LTL used here has facts and equalities over ground terms as atomic propositions, the usual propositional connectives (namely, $\neg$, $\vee$) and the temporal operators **X** (next), **F** (eventually) and **O** (once). Let π be an initialised path of M, an LTL formula ϕ *is valid on* π, written $\pi \models \phi$, if and only if $(\pi, 0) \models \phi$, where $(\pi, i) \models \phi$ (ϕ holds in π at time i) is inductively defined as:

$$
\begin{array}{lll}
(\pi, i) \models f & \text{iff} & f \in \pi(i) \ (f \text{ is a fact}) \\
(\pi, i) \models (t_1 = t_2) & \text{iff} & t_1 \text{ and } t_2 \text{ are the same terms} \\
(\pi, i) \models \neg\phi & \text{iff} & (\pi, i) \not\models \phi \\
(\pi, i) \models (\phi_1 \vee \phi_2) & \text{iff} & (\pi, i) \models \phi_1 \text{ or } (\pi, i) \models \phi_2 \\
(\pi, i) \models \mathbf{X}\phi & \text{iff} & (\pi, i+1) \models \phi \\
(\pi, i) \models \mathbf{F}\phi & \text{iff} & \exists j \in [i, \infty).(\pi, j) \models \phi \\
(\pi, i) \models \mathbf{O}\phi & \text{iff} & \exists j \in [0, i].(\pi, j) \models \phi
\end{array}
$$

In the sequel we use $(\phi_1 \wedge \phi_2)$, $(\phi_1 \Rightarrow \phi_2)$ and $\mathbf{G}\phi$ as abbreviations of $\neg(\neg\phi_1 \vee \neg\phi_2)$, $(\neg\phi_1 \vee \phi_2)$ and $\neg\mathbf{F}\neg\phi$ respectively.

The next section (Section 4.2) formalises the novel threat model by specifying the behaviour of principals using a set-rewriting formalism. This amounts to specifying the model M of the model checking problem (1). The subsequent section (Section 4.3) shows how interesting protocol properties can be formalised by means of LTL formulae, which correspond to G in the model checking problem. For what concerns C_I in the model checking problem (1), we will borrow the same LTL constraints that are detailed elsewhere [7] in order to capture assumptions

on communication channels. For example, an authentic channel *ch* whose input is intended to be exclusively accessible to a specified sender p can be captured as a conjunction of LTL constraints

$$\mathbf{G}(\texttt{sent}(rs, a, b, m, ch) \Rightarrow (a = p \wedge rs = p))$$

for each message m and agent b. Additional discussion would be out of the scope of this paper, while the interested reader may refer to [7].

4.2 Modelling Extensions

We conveniently adopt the IF language as it can specify inputs to the AVISPA backends and more specifically to SATMC, a successful SAT-based model checker [9] that will be used in the final validation phase. The following syntactical conventions are adopted in the sequel.

- Lower-case typewriter fonts, such as `na`, denote IF constants.
- Upper-case typewriter fonts, such as `A` and `KIDP`, denote IF variables.
- Lower-case italics fonts of 0 arity, such as s, compactly denote IF terms.
- Lower-case italics fonts of positive arity, such as $attack(a, v, s)$, denote meta-predicates that aim at improving the readability of this manuscript, but in fact do not belong to the current IF formalisation.
- Upper-case italics fonts serve two purposes: $VARS(t_1, \ldots, t_h)$ conveniently yields all IF variables occurring in terms $t_1, \ldots, t_h$; LHS yields the set of *positive* facts in the left hand side of a given rewriting rule.

To specify the MA threat model, a number of facts are necessary. Incidentally, facts in this section typically refer to IF terms because of the generality of the treatment. Some facts, enumerated in Table 1, already exist from previous work based on DY, and can be reused.

It is useful to expand on the three main facts that are reused here. If S is a set of facts representing a state, then the state of principal a is represented by the facts of form $\texttt{state}_r(j, a, es, s)$, called *state-facts*; es will be denoted by (a list of terms delimited by) square brackets (Section 5). It is assumed that for each session s and

Table 1 Existing facts and their informal meaning

Existing fact	Holds when
$\texttt{state}_r(j, a, es, s)$	Principal a, playing role r, is ready to execute step j in session s of the protocol, and has internal state es, which is the list of terms involved in her future steps.
$\texttt{c}(n)$	Term n is the current value of the counter used to issue fresh terms, and is incremented as $\texttt{s}(n)$ every time a fresh term is issued.
$\texttt{confidential}(m, g)$	Message m is confidential among the group of principals g.
$\texttt{sent}(rs, b, a, m, ch)$	Principal rs has sent message m on channel ch to principal a pretending to be principal b.
$\texttt{rcvd}(a, b, m, ch)$	Message m (supposedly sent by principal b) has been received on channel ch by principal a, but a has not processed it yet.
$\texttt{contains}(db, m)$	Message m is contained into set db. Sets are used, e.g., to share data between honest principals.

Table 2 New facts and their informal meaning

New fact	Holds when
$\texttt{nt}(a)$	Principal a is not trustworthy.
$\texttt{priv}(m, a)$	Message m is private for principal a.
$\texttt{ak}(a, m)$	Principal a knows message m.

for each principal a there exists at most one fact of the form $\texttt{state}_r(j, a, es, s)$ in S. This does not prevent a principal from playing different roles in different sessions.

The fact $\texttt{c}(n)$ holds of the current counter n used to generate fresh nonces. For example, $\texttt{c}(0)$ holds in the initial state, and $\texttt{c}(\texttt{s}(0))$ after the generation of the first fresh nonce, which takes the value 0. More generally, if the fact $\texttt{c}(\texttt{na})$ holds, a fresh nonce $\texttt{na}$ can be issued producing another state with counter $\texttt{c}(\texttt{s}(\texttt{na}))$.

The fact, $\texttt{confidential}(m, g)$ is normally used with the single m parameter in existing work based on DY, assuming that the only entity not meant to know it is the DY attacker. While DY reduces confidentiality of a message to keeping it confidential from the attacker, MA requires the original, subtler definition of confidentiality: "confidentiality is the protection of information from disclosure to those not intended to receive it" [34]. Thus, confidentiality of a message is compromised if ever anyone beyond its intended peers learns it. In consequence, both parameters are meaningful for us: the message that is stated confidential and the set of the only principals meant to know the message. Clearly, the set g is populated through occurrences of $\texttt{contains}(g, a)$ for each principal a intended to be in g.

The remaining facts in Table 1 are intuitive. A few more facts, summarised in Table 2, must be newly defined to reflect the MA threat model. Other new facts will be introduced later for the specific protocol under scrutiny.

All new facts deserve extra discussion. Under the MA threat model any principal may behave as a DY attacker. We may nonetheless need to formalise protocols that encompass trusted third parties, or where a principal loses her trustworthiness due to a certain event. The $\texttt{nt}(a)$ fact holds of a principal a that is not to be trusted. It may conveniently be used either statically, if added to the initial state of principals, or dynamically when introduced by rewriting rules. It is understood that if all principals but one are declared as trustworthy, then we are back to the DY threat model.

The fact $\texttt{priv}(m, a)$ is useful to implement MA, especially with respect to GA. It is stated for each principal in the protocol scenario under consideration and for each existing long-term secret of the principal, such as his long-term key in case of symmetric cryptography or his private-key in case of asymmetric cryptography.

The dedicated account on the attacker's knowledge in DY must be extended as a general account on principals' knowledge in MA. In practice, what was the $\texttt{ik}$ fact to represent the DY knowledge is now replaced by $\texttt{ak}$, which has as an extra parameter the principal's identity whose knowledge is being defined. Incidentally, we conveniently write down the set of facts in a state by enumerating the facts and interleaving them with a special dot. Here are the general rewriting rules defining $\texttt{ak}$, k and $\overline{k}$ being inverse keys of one another:

$$\texttt{ak}(a, \{m\}_k) \centerdot \texttt{ak}(a, \overline{k}) \xrightarrow{\texttt{decrypt}(VARS(a,\overline{k},m))} \texttt{ak}(a, m) \centerdot LHS$$

$$\texttt{ak}(a, \langle m_1, m_2 \rangle) \xrightarrow{\texttt{decompose}(VARS(a,m_1,m_2))} \texttt{ak}(a, m_1) \centerdot \texttt{ak}(a, m_2) \centerdot LHS$$

$$\texttt{ak}(a, m) \centerdot \texttt{ak}(a, k) \centerdot \neg\, \texttt{priv}(m, a) \xrightarrow{\texttt{encrypt}(VARS(a,k,m))} \texttt{ak}(a, \{m\}_k) \centerdot LHS$$

$$\texttt{ak}(a, m_1) \centerdot \texttt{ak}(a, m_2) \centerdot \neg\, \texttt{priv}(m_1, a) \centerdot \neg\, \texttt{priv}(m_2, a) \xrightarrow{\texttt{pairing}(VARS(a,m_1,m_2))} \texttt{ak}(a, \langle m_1, m_2 \rangle) \centerdot LHS$$

It can be observed that the `encrypt` and `pairing` rules only apply to messages that are not private for the generic principal. This enforces the requirement of the MA threat model that no principal ever reveals his long-term secrets to anyone. This restriction does not apply to short-term secrets such as session nonces, because they are not declared private.

The initial state of the system can then be defined easily in terms of state facts (`state`), knowledge facts (`ak`) and privacy facts (`priv`) for each principal participating in the protocol being studied—an example can be found below (Section 5.1.1). Once the initial state of the protocol is defined, the development of the protocol (through the execution of the steps it prescribes) is defined by a number of rewriting rules of the following two forms, respectively for the delivery and the sending of messages:

$$\texttt{sent}(rs, b, a, m, ch) \xrightarrow{\texttt{deliver}(VARS(rs,b,a,m,ch,))} \texttt{rcvd}(a, b, m, ch) \centerdot \texttt{ak}(a, m)$$

$$\texttt{rcvd}(a, b_1, m_1, ch_1) \centerdot \texttt{state}_r(j, a, es_1, s) \xrightarrow{\texttt{send}_i(VARS(a,b_1,b_2,rs,m_1,m_2,ch_1,ch_2,es_1,es_2,s))} \texttt{sent}(a, a, b_2, m_2, ch_2) \centerdot \texttt{state}_r(l, a, es_2, s) \centerdot \texttt{ak}(a, m_2)$$

The delivery rule models the reception of a message by an honest principal a supposedly from b when in fact the real sender is rs, whereas the sending rule models a's processing of a previously received message. More precisely, the sending rule states that if principal a playing role r is at step j in session s of the protocol and she has received message m_1 on channel ch_1 supposedly from b_1, then she can send message m_2 to b_2 on channel ch_2, store that message in her knowledge, and change her internal state accordingly preparing for step l.

Moreover, the sending rule will take slightly different forms depending on the protocol step it models. For example, if $j = 1$ and a sends the first message of the protocol, the fact $\texttt{rcvd}(a, b_1, m_1, ch_1)$ does not appear in the left hand side of the rule, reflecting a's freedom to initiate the protocol at anytime. Similarly, for generating and sending a fresh term, `c(N)` is included in the left hand side of the rule, while `c(s(N))` and $\texttt{ak}(a, \texttt{N})$ appear in the right hand side to express the incremented counter and the principal learning the fresh term. A further variant is necessary when the step involves either a membership test, an update of a set of elements, or a statement of data confidentiality. In this case, facts such as $\texttt{contains}(db, m)$ and $\texttt{confidential}(m, g)$ will be appropriately used (Section 5.1.1).

The malicious activity that any principal c may perform in MA is modelled rephrasing the standard rules for the DY attacker for a generic c as follows:

$$\texttt{nt}(c) \centerdot \texttt{ak}(c, m) \centerdot \texttt{ak}(c, a) \centerdot \texttt{ak}(c, b) \centerdot \texttt{ak}(c, ch) \centerdot \neg \texttt{priv}(m, c) \xrightarrow{\texttt{fake}(VARS(a,b,c,m,ch))} \texttt{sent}(c, a, b, m, ch) \centerdot LHS$$

$$\texttt{nt}(c) \centerdot \texttt{sent}(a, b_1, b_2, m, ch) \xrightarrow{\texttt{overhear}(VARS(a,b_1,b_2,c,m,ch))} \texttt{rcvd}(c, a, m, ch) \centerdot \texttt{ak}(c, m) \centerdot LHS$$

$$\texttt{nt}(c) \centerdot \texttt{sent}(a, b_1, b_2, m, ch) \xrightarrow{\texttt{intercept}(VARS(a,b_1,b_2,c,m,ch))} \texttt{rcvd}(c, a, m, ch) \centerdot \texttt{ak}(c, m) \centerdot \texttt{nt}(c)$$

The rule `fake` allows a non-trustworthy principal c to send a message impersonating another principal. The rule `overhear` allows a non-trustworthy principal c to learn a sent message, increasing their knowledge. The rule `intercept` allows a non-trustworthy principal c to learn a sent message, increasing their knowledge, and to remove the message from the channel.

Although the formalisation of message faking through rule `fake` is accurate, it does not yield the best model checking performance. In fact, the rule allows the forging of messages that will clearly not help to attack the protocol, as they do not correspond to any of the forms that the protocol prescribes. In other words, forging messages that no one will ever accept is of no use to any attacker, but expands the search space. The performance of the model checker improves by replacing that general rule with more specific ones, for example by introducing a forged message only if it resembles a protocol message. These are the impersonation rules, which are already available for DY [24, Section 3.2.4]. They have been first introduced in [28] where the authors show that a proper combination of *decomposing* (our decompose and decrypt), *diverting* (our intercept), and *impersonate* rules are sufficient to capture DY. The authors show that this optimisation preserves the existing attacks and does not introduce new ones. Impersonate rules can easily be recast in the MA threat model, as an example demonstrates below (Section 5.1.1), and the intuition behind their correctness applies also in MA.

Further modelling optimizations can be applied for simple protocols such as NSPK++. In there, the simplicity of the protocol lets us simplify channels away, and collapse the two general facts `sent` and `rcvd` into a single one, `msg`. In fact, distinguishing the action of having received a message (captured by `rcvd` facts) from that of having sent one (captured by `sent` facts) has been proven useful for us only to capture specific assumptions on communication channels (e.g., unilater SSL/TLS, resilient, authentic, etc). Protocols such as NSPK++ assume nothing on the communication channels (i.e., the channel is insecure) and thus do not require that distinction.[2] Receiving and sending a message by an agent becomes for those

[2]Notice that for these kind of protocols C_I would simply be the propositional true value.

protocols a single atomic transaction where the message is received and sent. As it is intuitive, $\mathtt{msg}(rs, b, a, m)$ signifies that principal rs, formalising the real sender, has sent message m to principal a pretending that m was sent by principal b. A rule modelling honest principals in NSPK++ is then of the form:

$$\mathtt{msg}(rs, b_1, a, m_1) \centerdot \mathtt{state}_r(i, a, es_1, s) \xrightarrow{\mathtt{step}_l(VARS(a,b_1,b_2,rs,es_1,es_2,m_1,m_2,s))} \mathtt{msg}(a, a, b_2, m_2) \centerdot \mathtt{state}_r(j, a, es_2, s) \centerdot \mathtt{ak}(a, m_1) \centerdot \mathtt{ak}(a, m_2)$$

where l is the protocol step number. In modelling a's receiving m_1 and sending out m_2, the rule demonstrates how to use existing and new facts together to model a protocol step under MA. It can be seen that the new state registers the facts $\mathtt{ak}(a, m_1)$ and $\mathtt{ak}(a, m_2)$. Also this rule, as the sending rule seen above, may take slightly different forms depending on the very protocol step it models. Of course, under this modelling optimization, also the rules capturing the malicious activity of principals have to be slightly adapted as follows:

$$\mathtt{nt}(c) \centerdot \mathtt{ak}(c, m) \centerdot \mathtt{ak}(c, a) \centerdot \mathtt{ak}(c, b) \centerdot \neg \mathtt{priv}(m, c) \xrightarrow{\mathtt{fake}(VARS(a,b,c,m))} \mathtt{msg}(c, a, b, m) \centerdot LHS$$

$$\mathtt{nt}(c) \centerdot \mathtt{msg}(a, b_1, b_2, m) \xrightarrow{\mathtt{overhear}(VARS(a,b_1,b_2,c,m))} \mathtt{ak}(c, m) \centerdot LHS$$

$$\mathtt{nt}(c) \centerdot \mathtt{msg}(a, b_1, b_2, m) \xrightarrow{\mathtt{intercept}(VARS(a,b_1,b_2,c,m))} \mathtt{ak}(c, m) \centerdot \mathtt{nt}(c)$$

4.3 Validation Extensions

One may expect that standard security properties such as confidentiality and authentication can be routinely specified and checked under the MA threat model using a model checker. Yet, confidentiality deserves particular consideration in what the MA model differs from the DY one.

In general, the confidentiality of a message m w.r.t. a group of principals g is guaranteed if and only if m is only known to principals in g. This property is clearly violated anytime a principal a outside g learns, for any reason, m. However, under DY, where all principals but the attacker meticulously follow the protocol, the violation is by design not considered so unless a is the attacker. There are many real scenarios—ranging from a betrayed person who publishes his/her partner's credit card details, to a scenario where a fired employee discloses sensitive information about its former employer—in which this violation is significant and therefore not negligible. In the MA model this confidentiality breach is not overlooked. It can be

checked by the following meta-predicate formalising a violation of confidentiality of a message m by a principal a who is not in the group of principals g meant to know m:

$$voc(a, m, g) = \mathbf{F}(\texttt{confidential}(m, g) \wedge \texttt{ak}(a, m) \wedge \neg\, \texttt{contains}(g, a))$$

The negation of this meta-predicate corresponds to G in the model checking problem stated in Section 4.1, and represents the confidentiality property.

What is more interesting is that the MA threat model paves the way to investigate subtler protocol properties than confidentiality and authentication. We already observed (Section 3) that retaliation is common in the real world. (Even anyone who is most peaceful may react under attack—whether this is fortunate or unfortunate lies outside our focus.) Depending on who reacts against the attacker, retaliation can be named differently, according to a previous publication [18]: if it is the victim, then there is *direct retaliation*; if it is someone else, then there is *indirect retaliation*. However, as we shall see in the sequel of this section, retaliation may also take place in a third scenario, which is unpublished.

Let $attack(c, a, b)$ be a meta-predicate holding if and only if an attacker c has successfully attacked a victim a (being $a \neq c$ and $a \neq b$) with the (unaware) support of b (if any). Of course, "has successfully attacked" denotes the violation of the specific property that the protocol under analysis is supposed to achieve. Any violation of a protocol property should make the predicate hold, and therefore the predicate should be defined by cases. For example, we shall see below (Section 5.1.2) how the meta-predicate is defined in terms of various illegal money transfers.

Once an attack is defined, various retaliation scenarios can be defined over two of its instances as the following meta-predicates:

$$direct\_retaliation\_scenario(a, c, b_1, b_2) = \mathbf{F}(attack(c, a, b1) \wedge \mathbf{X}\,\mathbf{F}\, attack(a, c, b2))$$

$$indirect\_retaliation\_scenario(a, c, b) = \mathbf{F}(attack(c, a, b) \wedge \mathbf{X}\,\mathbf{F}\, attack(b, c, a))$$

$$conjuring\_retaliation\_scenario(a, c, b) = \mathbf{F}(attack(c, a, b) \wedge \mathbf{X}\,\mathbf{F}\, attack(b, a, c))$$

It can be seen that the meta-predicate for a direct retaliation scenario is valid on those paths where an attacker hits a victim who hits the attacker back. Each attack can be carried out with the help of potentially different supporters $b1$ and $b2$. By contrast, the meta-predicate for an indirect retaliation scenario shows that who hits the attacker back is not the victim but the supporter, who perhaps realises what he has just done and decides to rebel against the attacker. Finally, a conjuring retaliation scenario, which has never been formalised before, holds when an attack against a victim is then exploited by the previous supporter against the same victim; it may then be interpreted as a form of collusion.

A new type of attack scenario, which we call *anticipation* is possible. The corresponding informal analysis was not given above (Section 3) to reflect the timeline of our research: the anticipation scenario was not discovered by informal analysis and then validated but, rather, our validation experiments yielded it. Therefore, only the gist of anticipation is presented here, while its detailed explanation will follow (Section 5.1.2).

An anticipation scenario is that of an attack initiated and not yet finalised by a principal, but exploited by someone else. This is realistic at present, when a successful attack, instead of consisting of a single move, is an ordered sequence of required

moves. In consequence, using a self-explaining meta-predicate for the initialisation of the attack, the anticipation scenario can be defined as:

$$anticipation\_scenario(c, a, b) = \mathbf{F}(attack\_init(c, a, b) \wedge \mathbf{X}\,\mathbf{F}\, attack(b, c, a))$$

As it is customary, the $attack\_init(c, a, b)$ meta-predicate will meet a specific definition over an example analysis (Section 5.1.2).

5 Validating Protocols Under MA

This section reports our mechanised validation of the previous informal analysis (Section 3) of a classical protocol (Section 5.1) and a modern deployed protocol (Section 5.2).

5.1 A Classical Protocol

5.1.1 Model Outline

Let us illustrate a portion of the specification of the NSPK++ protocol (Section 3.1). The initial state for a protocol scenario featuring three principals `a`, `b` and `c`, with the first playing the role of initiator (also said alice) can easily be defined. The portion of initial state concerning `a` (the rest is omitted for brevity) is as follows:

$$\begin{aligned}
&\mathtt{nt(a).nt(b).nt(c).}\\
&\mathtt{ak(a,a).ak(a,b).ak(a,c).}\\
&\mathtt{ak(a,pk(ka)).ak(a,pk(kb)).ak(a,pk(kc)).}\\
&\mathtt{ak(a,inv(pk(ka))).}\\
&\mathtt{priv(inv(pk(ka)),a).}\\
&\mathtt{state}_{\mathtt{alice}}\mathtt{(0,a,[c,ka,kc,set_ac],1).state}_{\mathtt{alice}}\mathtt{(0,a,[b,ka,kb,set_ab],2)}
\end{aligned}$$

The rewriting rules modelling the protocol steps feature IF variables so that they apply to any principals in the scenario that is defined. For example, the step in which `B` receives the first message of the protocol (supposedly) from `A` and replies with the second protocol message is modelled by the following rule:

$$\begin{aligned}
&\mathtt{msg(RS,A,B,\{A,NA\}_{KB}).state}_{\mathtt{bob}}\mathtt{(1,B,[A,KA,KB,G],S).c(NB)}\\
&\qquad\xrightarrow{\mathtt{step}_2\mathtt{(B,A,RS,KA,KB,NA,NB,G,S)}}\\
&\mathtt{msg(B,B,A,\{NA,NB\}_{KA}).state}_{\mathtt{bob}}\mathtt{(3,B,[A,KA,KB,NA,NB,G],S).}\\
&\mathtt{ak(B,\{A,NA\}_{KB}).ak(B,NB).contains(G,A).contains(G,B).}\\
&\mathtt{confidential(NB,G).c(s(NB))}
\end{aligned}$$

where `G` represents the group of principals that are allowed to share the freshly generated nonce `NB`.

To provide another example, let us consider the completion step $4b$ in which `A` receives and then executes a money transfer from `B`. We introduce the fact $\mathtt{transferred}(rs, a, b, c)$ to formalise rs who, in the disguise of a, transferred some money (the very amount is abstracted away) from a's account to c's at b (which intuitively is a bank) in session s. Another fact is needed, that is $\mathtt{belief}(a, b, m)$, to

formalise that principal a believes message m is associated to principal b. After the three standard steps of the NSPK protocol (Fig. 1), both participants state what they believe: in particular, belief(A, B, ⟨NA, NB⟩) is generated when A sends message 3 and belief(B, A, ⟨NA, NB⟩) when B receives that message. The protocol step 4b is then modelled by two rules, one for B's sending and one for A's reception. Here is the latter:

$$\texttt{msg(RS, B, A, \{tr(B, C, X)\}}_{\texttt{⟨NA,NB⟩}}\texttt{)} \bullet \texttt{belief(A, B, ⟨NA, NB⟩)}$$
$$\xrightarrow{\texttt{step}_{4\_\texttt{rec}}\texttt{(A,B,C,RS,NA,NB,X)}}$$
$$\texttt{ak(A, \{tr(B, C, X)\}}_{\texttt{⟨NA,NB⟩}}\texttt{)} \bullet \texttt{ak(A, X)} \bullet \texttt{transferred(RS, B, A, C)}$$

The rule states the fact transferred(RS, B, A, C), which is not to be confused with $\{\texttt{tr(B, C, X)}\}_{\texttt{⟨NA,NB⟩}}$ — the latter is a message expressing the request of a transfer. This formalisation simplifies a published version [11] thanks to the newly introduced fact belief(A, B, ⟨NA, NB⟩). The current rule is also more general in that it applies to the reception of step 4a too. Therefore, one pair (rather than two pairs) now suffices to model the two completion steps.

As mentioned in the general treatment (Section 4.2), the fake rule is in practice replaced by a number of impersonation rules. For brevity, we only quote one such rule here. A principal C who, pretending to be A, sends the first message of our example protocol to B, who is exactly waiting for a message of that form, is modelled as:

$$\texttt{nt(C)} \bullet \texttt{state}_{\texttt{bob}}\texttt{(2, B, [A, KA, KB, G], S)} \bullet \texttt{ak(C, A)} \bullet \texttt{ak(C, NA)} \bullet \texttt{ak(C, KB)}$$
$$\xrightarrow{\texttt{impersonate}_2\texttt{(C,A,B,KA,KB,NA,G,S)}}$$
$$\texttt{msg(C, A, B, \{A, NA\}}_{\texttt{KB}}\texttt{)} \bullet \texttt{ak(C, \{A, NA\}}_{\texttt{KB}}\texttt{)} \bullet \mathit{LHS}$$

5.1.2 Validation

In running SATMC over the formalisation of the NSPK++ protocol, we considered the classical scenario in which a wants to talk with b in one session and with c in another session. Because the formalisation accounted for the new threat model, we were pleased to observe that it passed simple sanity checks: SATMC outputs the known confidentiality attack whereby c learns nb, and the known authentication attack whereby c impersonates a with b. As these are well known, they are omitted here. More importantly, the tool also reported what are our major findings: b's confidentiality attack upon na, and interesting retaliation attacks, which are detailed in the following. We argue that this is the first mechanised treatment of these subtle properties, laying the ground for much more computer-assisted analysis of security protocols.

We set off by reporting the protocol traces that the tool finds when checking a violation of confidentiality by the meta-predicate $voc(a, m, g)$ already defined (Section 4.3). Each trace is obtained by mildly polishing the partial-order plan returned by SATMC. Each trace element reports the label of the rule that fired, and hence the trace can be easily interpreted as the history of events leading to the confidentiality attack.

Figure 6 shows the trace obtained by running SATMC with the formula voc(b, na, set_ac), where set_ac = {a, c}.

```
0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
1: [overhear(c,a,a,c,{na,a}_kc)]
2: [decrypt(c,inv(kc),{na,a})]
3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
4: [step_2(b,a,c,ka,kb,na,nb,set_ab,2)]
5: [decrypt(b,inv(kb),{na,a})]
```

Fig. 6 Finding b's confidentiality attack upon na

A full description of the trace requires a glimpse at the protocol formalisation—each rule label in a trace element must be matched to the actual rule—but we shall see that the trace can be automatically converted into a more user-friendly version. Element 0 means that principal a initiates the protocol with principal c. Elements 1, 2 and 3 show c's illegal activities respectively of getting the message, decrypting it and forging a well-formed one for principal b. Although c does not need in practice to overhear a message meant for him, element 1 is due to our monolithic formalisation of the acts of receiving a message and of sending out its protocol-prescribed reply. Therefore, for a principal to abuse a message, the principal must first overhear it. Element 3 reminds that c's forging of the message for b counts as an attempt to impersonate a. The last two elements signal b's legal participation in the protocol by receiving the message meant for him, and his subsequent deciphering of the nonce. SATMC now returns because $voc(\mathtt{b}, \mathtt{na}, \mathtt{set\_ac})$ holds (where the set $\mathtt{set\_ac} = \{\mathtt{a}, \mathtt{c}\}$).

Of course, the tool also finds the classical confidentiality attack by c upon nb. Figure 7 presents the output when checking $voc(\mathtt{c}, \mathtt{nb}, \mathtt{set\_ab})$ with $\mathtt{set\_ab} = \{\mathtt{a}, \mathtt{b}\}$. The trace can be easily understood as a continuation of the one in Fig. 6 with the four extra elements showing c's malicious intervention. Precisely, elements 6 and 7 show that b is impersonating c with a, who naively replies to c. The last two elements confirm that c gets the message from which he can decrypt the nonce nb. Also, it can be seen that the impersonation rules can be applied by both b and c, reflecting the MA threat model.

Having validated the confidentiality issues, we can move on to studying retaliation. It is useful to define an illegal money transfer in terms of the appropriate fact:

$$illegal\_transfer(c, a, b) = \mathtt{transferred}(c, a, b, c) \ \wedge \ c \neq a$$

It can be seen that the transfers that are illegal are only those where the requester differs from the source account holder. For simplicity, the requester is defined to

```
0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
1: [overhear(c,a,a,c,{na,a}_kc)]
2: [decrypt(c,inv(kc),{na,a})]
3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
4: [step_2(b,a,c,ka,kb,na,nb,set_ab,2)]
5: [decrypt(b,inv(kb),{na,a})]
6: [impersonate_3(b,c,a,ka,kc,na,nb,set_ac,1)]
7: [step_3(a,c,b,ka,kc,na,nb,set_ac,1)]
8: [overhear(c,a,a,c,{nb}_kc)]
9: [decrypt(c,inv(kc),{nb})]
```

Fig. 7 Finding c's confidentiality attack upon nb

```
     ...
10: [impersonate_3_rec(c,a,b,ka,kb,nb,set_ab)]
11: [step_3_rec(b,a,c,ka,kb,na,nb,set_ab,2)]
12: [impersonate_4_rec(c,a,b,c,na,nb)]
13: [step_4_rec(b,a,c,c,na,nb)]
14: [impersonate_4_rec(b,c,a,b,na,nb)]
15: [step_4_rec(a,c,b,b,na,nb)]
```

Fig. 8 Finding an indirect retaliation scenario

equal the target account holder. Because the attack of interest here is exactly an illegal money transfer, we define:

$$attack(c, a, b) = illegal\_transfer(c, a, b)$$

The model checker can now be run on *indirect_retaliation_scenario*(*a*, *c*, *b*), which was defined in terms of two attacks (Section 4.3). The tool returns the trace that continues the one in Fig. 7 as shown in Fig. 8. Element 10 formalises c's impersonation of a with b, while the subsequent element confirms that b replies by taking a legal step. Elements 12 and 13 then identify c's attack. The latter in particular indicates a firing of the rule that makes *attack*(c, a, b) hold. The last two elements witness b's retaliation attack by making *attack*(b, c, a) hold. Therefore, the definition of indirect retaliation scenario(Section 4.3) is met, that is *indirect_retaliation_scenario*(a, c, b) holds, indicating that the tool confirms the retaliation analysis done above (Section 3.1).

A graphical illustration of this retaliation scenario is in Fig. 9, and can be easily built. First, we run a procedure to transform the output of the tool into a more readable and intuitive version. Then, we coherently associate the significant phrases

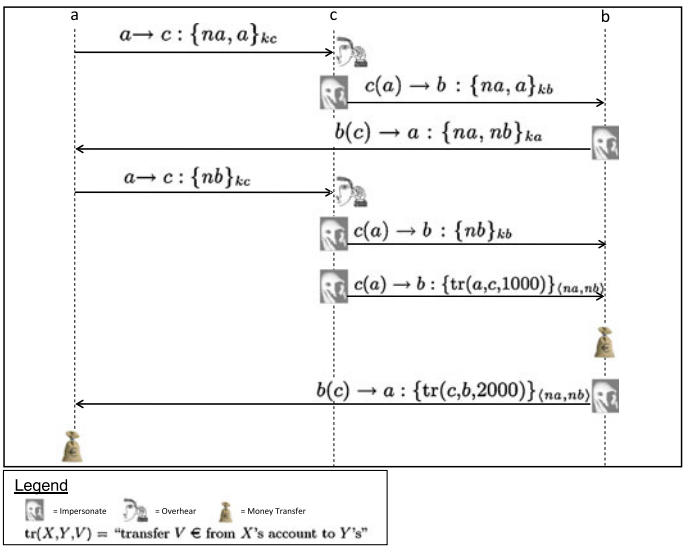


Fig. 9 Graphics of b's indirect retaliation

of this version to graphical elements, and hence build the image. The behaviours and interactions of the principals can be observed by looking at the figure from top to bottom. The overhear and impersonate icons emphasise the significant number of illegal steps taken by both `c` and `b`.

We now turn our attention to validating all six possible attack attempts studied above in isolation (Section 3.1) rather than paired up into retaliation scenarios. The fact $\texttt{belief}(a, b, \langle na, nb \rangle)$ formalising b's association of the nonce pair $\langle na, nb \rangle$ to a (Section 5.1.1) is useful here. It lets us build a simple meta-predicate to formalise Lowe's scenario in terms of beliefs of principals:

$$\begin{aligned} \mathit{lowe\_scenario}(a, b, c, na, nb) = {} & \texttt{belief}(a, c, \langle na, nb \rangle) \\ & \wedge \neg\texttt{belief}(a, b, \langle na, nb \rangle) \\ & \wedge \texttt{belief}(b, a, \langle na, nb \rangle) \\ & \wedge \neg\texttt{belief}(b, c, \langle na, nb \rangle) \\ & \wedge \texttt{belief}(c, a, \langle na, nb \rangle) \\ & \wedge \texttt{belief}(c, b, \langle na, nb \rangle) \end{aligned}$$

All attack attempts (Section 3.1) can then be modelled in a coherent form, including Lowe's attack [12] and its indirect retaliation attack [11], that is as a further specification of the meta-predicate just defined. Therefore, the meta-predicate $\mathit{attack}(c, a, b)$ introduced for protocol attacks (Section 4.3) is refined as the following six meta-predicates:

$$\begin{aligned} \mathit{attack\_I}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(c, a, b)\,) \\ \mathit{attack\_II}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(b, c, a)\,) \\ \mathit{attack\_III}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(a, c, b)\,) \\ \mathit{attack\_IV}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(b, a, c)\,) \\ \mathit{attack\_V}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(c, b, a)\,) \\ \mathit{attack\_VI}(a, b, c, na, nb) &= \mathbf{F}(\, \mathit{lowe\_scenario}(a, b, c, na, nb) \\ &\quad \wedge\ \mathbf{X}\,\mathbf{F}\, \mathit{illegal\_transfer}(a, b, c)\,) \end{aligned}$$

These are the properties to run the tool against. To validate attack I, we launched the tool with the formula $\mathit{attack\_I}(a, b, c, na, nb)$ and obtained as output the partial order plan that appears polished in Fig. 10.

The trace should be read bearing in mind that the sets `set_ac` and `set_ab` have the conventional definitions already seen above. It can be appreciated that the

```
 0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
 1: [overhear(c,a,a,c,{na,a}_kc)]
 2: [decrypt(c,inv(kc),{na,a})]
 3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
 4: [step_2(b,a,c,ka,kb,na,nb,set_ab,2)]
 5: [decrypt(b,inv(kb),{na,a})]
 6: [impersonate_3(b,c,a,ka,kc,na,nb,set_ac,1)]
 7: [step_3(a,c,b,ka,kc,na,nb,set_ac,1)]
 8: [overhear(c,a,a,c,{nb}_kc)]
 9: [decrypt(c,inv(kc),{nb})]
10: [impersonate_3_rec(c,a,b,ka,kb,nb,set_ab)]
11: [step_3_rec(b,a,c,ka,kb,na,nb,set_ab,2)]
12: [impersonate_4_rec(c,a,b,c,na,nb)]
13: [step_4_rec(b,a,c,c,na,nb)]
```

Fig. 10 Finding attack I, Lowe's attack

trace is the expected continuation of the one in Fig. 7. It is thus the same as the indirect retaliation trace in Fig. 8 pruned of its last two elements. This provides extra clarification of retaliation as a pair of attacks, whereas we are currently investigating all possible instances of a single attack.

We also validated attack attempt II by running SATMC over the formula $attack\_II(a, b, c, na, nb)$. The output was a trace that only differs from the previous attack trace in the last two elements, which are given in Fig. 11. This trace is therefore equal to the indirect retaliation trace of Fig. 8 pruned of its elements 12 and 13. It is then formally confirmed that the indirect retaliation found above pairs up attacks I and II.

Another test was to run the tool over $attack\_IV(a, b, c, na, nb)$. The output confirmed our expectations, being a trace identical to that in Fig. 10 except for the last two elements, given in Fig. 12, which express `b`'s stealing from `a`'s account at `c`. Precisely, the elements as usual report the labels of the rules that state the facts behind the target meta-predicate. In particular, the last element says that rule $\texttt{step}_{4\_\texttt{rec}}$ (Section 5.1.1) was applied with specific parameters, thus stating exactly the illegal money transfer that defines the target attack.

A similar finding derived from running the model checker over the formula $attack\_VI(a, b, c, na, nb)$. It reported a trace that only deviates from the one in Fig. 10 for the last two elements, pictured in Fig. 13. They indicate in the usual style that `b` is stealing from `a`'s account at `c`.

When the tool checked $attack\_III(a, b, c, na, nb)$, it stopped reporting no attacks. The same outcome was obtained with $attack\_V(a, b, c, na, nb)$. Also, when checking $direct\_retaliation\_scenario(a, c, b, b)$, the tool output a trace that continues that for attack IV with the last two elements of that for attack VI: the two attacks clearly form

```
    ...
12: [impersonate_4_rec(b,c,a,b,na,nb)]
13: [step_4_rec(a,c,b,b,na,nb)]
```

Fig. 11 Finding attack II, indirect retaliation attack of Lowe's

```
    ...
12: [impersonate_4_rec(b,a,c,b,na,nb)]
13: [step_4_rec(c,a,b,b,na,nb)]
```

Fig. 12 Finding attack IV, our attack

a direct retaliation scenario. The same consideration applies to *conjuring_retaliation_scenario*(a, c, b) concerning attacks I and IV. Therefore, we conclude that SATMC always confirmed our informal analysis (Section 3.1).

We have seen that all attack attempts can be modelled and validated coherently, that is as attempts to carry out illegal money transfers upon Lowe's man-in-the middle scenario. It seems that the extra burden that the MA threat model causes upon the model checker is manageable.

This completes the validation phase of our informal analysis of the protocol (Section 3.1). However, we also run a variety of subsidiary tests to obtain extra insights, each time adjusting the property to check. In particular, we tried an alternative definition of the indirect retaliation scenario (Section 4.3) where the **X** operator was left out. Somewhat to our surprise, the tool returned a trace leading to a state where both attacks held, as if retaliation did not need be triggered by another attack. This called for more attention at the tool output.

It was exactly as a consequence of this test that we observed from element 3 in Fig. 7 that `c` initialises his attack very early—precisely, with his impersonation of `a` based upon the repetition to `b` of `a`'s nonce `na`. This means that `b` learns `na` when `c` has not yet learned `nb` and hence cannot attack yet. To clarify and validate this intuition, we defined the self-explaining fact `nonce_leak`(c, a, b).

In the MA threat model, it is plausible that `b` exploits his knowledge before `c` does. This can be interpreted as a scenario in which a potential victim realises that he is going to be attacked, and therefore reacts successfully before being actually attacked. We name this *anticipation scenario* and define it as a meta-predicate below. To improve performance, we decided to add a rule that introduces the fact `nonce_leak(C, A, B)` following the protocol step that initialises the attack:

$$\texttt{confidential(NA, G)}.\texttt{ak(B, NA)}.\neg\texttt{contains(G, B)}.\ \texttt{ak(B, KB}^{-1}\texttt{)}.\texttt{msg(C, A, B, \{A, NA\}}_{\texttt{KB}}\texttt{)} \xrightarrow{\texttt{i\_got\_you(C,A,B,NA,G)}} \texttt{nonce\_leak(C, A, B)}.LHS$$

A meta-predicate *attack_init*(c, a, b) must be introduced to formalise some principal c's act of initialising an attack, which is yet to be carried out, while interleaving

```
    ...
12: [impersonate_4_rec(a,b,c,a,na,nb)]
13: [step_4_rec(c,b,a,a,na,nb)]
```

Fig. 13 Finding attack VI, direct retaliation attack of our attack

sessions with some a and b. In the same fashion as with the attack meta-predicate, we provide a definition that is appropriate for the attack under analysis, and corresponds to the nonce leak:

$$attack\_init(c, a, b) = \texttt{nonce\_leak}(c, a, b)$$

SATMC provided the expected output. When the tool was run on the goal *anticipation_scenario*(c, a, b), which was defined above (Section 4.3), it produced the trace reported in Fig. 14.

This trace is not difficult to understand with the experience gained through the previous experiments. It can be observed that it develops up to element 5 as in the previous figures, that is up to the point that b gets nonce na. Element 6 indicates the firing of the rule just presented, which signals the initialization of the attack making *attack_init*(c, a, b) hold. The next two elements are already known, as they indicate that the scenario develops till a takes the third step of the protocol with c. The last two elements show that before c decrypts the message and finds nb (as formalised for example by elements 8 and 9 in Fig. 7 or Fig. 10), principal b can successfully impersonate c with a. The last element in particular makes *attack*(b, c, a) hold, so that also *anticipation_scenario*(c, a, b) holds. It may be useful to read the development of this scenario from the graphical illustration in Fig. 15, which was constructed analogously to the previous one.

Our various experiments produced another interesting finding. When we were checking *direct_retaliation_scenario*(b, c, a, a), SATMC reported a trace describing a scenario that may occur under the MA threat model. After a session with b, principal a discloses to c the nonce pair of the session. After another session with c, principal a discloses to c the nonce pair of the session. In consequence, both b and c become capable of attacking each other with a's support (for example, at bank a). This reflects the real-world situation in which someone creates strife in a couple that starts fighting, or more simply a bank's misuse of its customers' credentials, which has not been rare [4]. To the best of our knowledge, this scenario has never been formally analysed before. The corresponding trace (omitted here) shows that a uses an impersonation rule to disclose each nonce pair to the other principal. This is

```
 0: [step_1(a,c,ka,kc,na,nb,set_ac,1)]
 1: [overhear(c,a,a,c,{na,a}_kc)]
 2: [decrypt(c,inv(kc),{na,a}_kc)]
 3: [impersonate_2(c,a,b,ka,kb,na,set_ab,2)]
 4: [step_2(b,a,c,ka,kb,na,nb,set_ab,2)]
 5: [decrypt(b,inv(kb),{na,a}_kb)]
 6: [i_got_you(c,a,b,na,set_ac)]
 7: [impersonate_3(b,c,a,ka,kc,na,nb,set_ac,1)]
 8: [step_3(a,c,b,ka,kc,na,nb,set_ac,1)]
 9: [impersonate_4_rec(b,c,a,b,na,nb)]
10: [step_4_rec(a,c,b,b,na,nb)]
```

Fig. 14 Finding b's anticipation attack

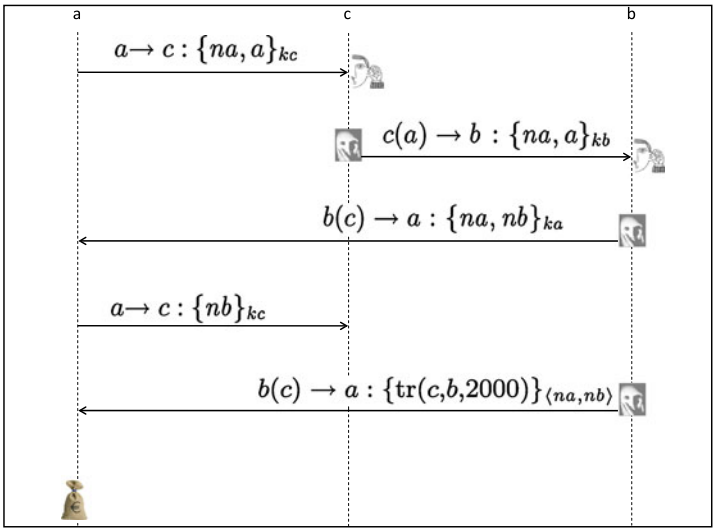


Fig. 15 Graphics of b's anticipation attack

possible because the MA threat model only constrains each principal to retain long-term secrets, but not short-term, session secrets.

5.2 A Modern Deployed Protocol

5.2.1 Model Outline

Hereafter we focus on modelling the Google SSO protocol under the MA threat model. This amounts to adapting the model of the Google SSO protocol detailed in [7] in the MA setting. Not surprisingly, the Google SSO protocol is more complex to model than NSPK++. Because of its assumptions on the communication channels, the rewriting rules are given in terms of `sent` and `rcvd` facts, which feature a parameter formalising the channel (Section 4.2). For example, the protocol step that sees a service provider `SP` replying with an authentication request (second message of Fig. 4) is modelled as follows:

$$\begin{aligned}
&\mathtt{rcvd}(\mathtt{SP}, \mathtt{C}, \langle \mathtt{C}, \mathtt{SP}, \mathtt{URI}\rangle, \mathtt{CHANC2SP}).\\
&\mathtt{state}_{\mathtt{sp}}(0, \mathtt{IDP}, \mathtt{C}, [\mathtt{KIDP}, \mathtt{URI}], \mathtt{S}).\mathtt{c}(\mathtt{ID})\\
&\qquad \xrightarrow{\mathtt{step_sp}_{SIA1}(\mathtt{SP},\mathtt{IDP},\mathtt{C},\mathtt{URI},\mathtt{ID},\mathtt{KIDP},\mathtt{CHANSP2C},\mathtt{CHANC2SP},\mathtt{S})}\\
&\mathtt{sent}(\mathtt{SP}, \mathtt{SP}, \mathtt{C}, \langle \mathtt{IDP}, \mathtt{SP}, \mathtt{ID}, \mathtt{URI}\rangle, \mathtt{CHANSP2C}).\mathtt{ak}(\mathtt{SP}, \mathtt{ID}).\\
&\mathtt{ak}(\mathtt{SP}, \langle \mathtt{IDP}, \mathtt{SP}, \mathtt{ID}, \mathtt{URI}\rangle).\\
&\mathtt{state}_{\mathtt{sp}}(1, \mathtt{IDP}, \mathtt{C}, [\mathtt{KIDP}, \mathtt{URI}, \mathtt{ID}], \mathtt{S}).\mathtt{c}(\mathtt{s}(\mathtt{ID}))
\end{aligned}$$

This rule differs from the corresponding one presented in [7] for the only addition of `ak` in the right-hand-side.

Similarly, the final step where a service provider SP makes available at the fresh address resource(SP, N) what C originally requested (last message of Fig. 4) can be formalised as follows:

$$\begin{aligned}
&\mathtt{rcvd}(\mathtt{SP}, \mathtt{C}, \langle \{\mathtt{C}, \mathtt{IDP}\}_{\mathtt{inv(KIDP)}}, \mathtt{URI}\rangle, \mathtt{CHANC2SP}).\\
&\mathtt{state}_{\mathtt{sp}}(1, \mathtt{IDP}, \mathtt{C}, [\mathtt{KIDP}, \mathtt{URI}, \mathtt{ID}], \mathtt{S}).\mathtt{c}(\mathtt{N})\\
&\quad \xrightarrow{\mathtt{step_sp}_{A4S2}(\mathtt{SP},\mathtt{IDP},\mathtt{C},\mathtt{URI},\mathtt{N},\mathtt{ID},\mathtt{KIDP},\mathtt{CHANSP2C},\mathtt{CHANC2SP},\mathtt{S})}\\
&\mathtt{sent}(\mathtt{SP}, \mathtt{SP}, \mathtt{C}, \mathtt{resource}(\mathtt{SP}, \mathtt{N}), \mathtt{CHANSP2C}).\mathtt{ak}(\mathtt{SP}, \mathtt{N}).\\
&\mathtt{ak}(\mathtt{SP}, \mathtt{resource}(\mathtt{SP}, \mathtt{N})).\mathtt{contains}(\mathtt{ac}, \langle \mathtt{resource}(\mathtt{SP}, \mathtt{N}), \mathtt{C}\rangle).\\
&\mathtt{state}_{\mathtt{sp}}(2, \mathtt{IDP}, \mathtt{C}, [\mathtt{KIDP}, \mathtt{URI}, \mathtt{ID}, \mathtt{N}], \mathtt{S}).\mathtt{c}(\mathtt{s}(\mathtt{N}))
\end{aligned}$$

where ac is an overall access control set containing associations between resource addresses provided by a specific service provider and clients allowed to access them.

For each of these honest participation rules, we also have an impersonate rule in the same style seen for the NSPK++ model (Section 5.1.1) to improve efficiency with respect to using the general fake rule (Section 4.2).

5.2.2 Validation

For validation we have used the scenario where bob is a client of two service providers, called google and i. bob is honest while google and i can both be malicious. The tool reported our main findings, that the confidentiality attack of i can be retaliated by a confidentiality attack of google (Section 3.2).

The attack meta-predicate was defined as:

$$\begin{aligned}
\mathit{attack}(i, c, sp) = {} & \mathtt{contains}(\mathtt{ac}, \langle \mathtt{resource}(sp, n), c\rangle) \wedge\\
& \mathtt{ak}(i, \mathtt{resource}(sp, n)) \wedge\\
& \neg\, \mathtt{contains}(\mathtt{ac}, \langle \mathtt{resource}(sp, n), i\rangle) \wedge\\
& i \neq sp
\end{aligned}$$

We ran the model checker on *conjuring_retaliation_scenario*(c, i, sp) to check whether the original Google attack can be followed by a second attack against bob, where the original attack is formalised as *attack*(i, bob, google) and the second as *attack*(google, bob, i). The answer is it can, a polished version of the trace is shown in Fig. 16.

The trace can be understood as follows. Element 0 means that the client bob initiates the protocol with the service provider i. Elements 1 to 4 show i's activities respectively of getting the message, decomposing it and forging a well-formed answer for the client bob, who receives it. Elements 5 to 8 show the exchange between bob and the identity provider idp, where bob requests and receives the authentication assertion. Element 9 shows two activities in parallel: i initiates a session with the service provider google impersonating bob; and bob forwards the authentication request to i. Elements 11 to 15 show the continuation of i's illegal activities, where i keeps impersonating bob with google, and these end with i's learning the resource of bob at google. The original Google attack is now completed, and element 16 shows the detection rule, which fires exactly when *attack*(i, bob, google) hold, and serves as an indicator of the first attack. Note that the detection rule has

```
 0: [step_c_S1(bob,i,uri,c2spA,1)]
 1: [deliver(bob,bob,i,bob.i.uri,c2spA)]
 2: [decompose_1(i,bob,i,uri)]
 3: [impersonate_1(i,bob,i,idp,uri,anyid,sp2cA,1)]
 4: [deliver(i,i,bob,idp.i.anyid.uri,sp2cA)]
 5: [step_c_A1A2(bob,i,idp,uri,anyid,c2idpA,sp2cA,1)]
 6: [deliver(bob,bob,idp,idp.i.anyid.uri,c2idpA)]
 7: [step_idp_A2A3(idp,i,bob,idp2cA,c2idpA,2,anyid,uri,kidp)]
 8: [deliver(idp,idp,bob,i.{bob.idp}_inv(kidp).uri,idp2cA)]
 9: [impersonate_6(i,google,idp,bob,calendar,c2spB,3),
     step_c_A3A4(bob,i,idp,uri,anyid,idp2cA,c2spA,1)]
10: [deliver(bob,bob,i,i.{bob.idp}_inv(kidp).uri,c2spA),
     deliver(i,bob,google,bob,bob.google.calendar,c2spB)]
11: [decompose_3(i,idp,kidp,bob,uri,i),
     step_sp_S1A1(google,idp,bob,calendar,0,kidp,sp2cB,c2spB,3)]
12: [impersonate_7(i,google,idp,bob,calendar,id,c2spB,3)]
13: [deliver(i,bob,google,google.{bob.idp}_inv(kidp).calendar,c2spB)]
14: [step_sp_A4S2(google,idp,bob,calendar,s(0),id,kidp,sp2cB,c2spB,3)]
15: [overhear(i,sp2cB,google,bob,resource(google,s(0)))]
16: [detect]
17: [impersonate_6(google,i,idp,bob,mail,c2spC,7)]
18: [deliver(google,bob,i,bob.i.mail,c2spC)]
19: [step_sp_S1A1(i,idp,bob,mail,s(s(0)),kidp,sp2cC,c2spC,7)]
20: [impersonate_7(google,i,idp,bob,mail,idi,c2spC,7)]
21: [deliver(google,bob,i,i.{bob.idp}_inv(kidp).mail),c2spC)]
22: [step_sp_A4S2(i,idp,bob,mail,s(s(s(0))),idi,kidp,sp2cC,c2spC,7)]
23: [overhear(google,sp2cC,i,bob,resource(i,s(s(s(0)))))]
```

Fig. 16 Finding a conspiring retaliation scenario

no parameters because it does not feature variables but only constants. Then, elements 17 to 23 show the retaliation of `google`, who completes a session with the service provider `i`, impersonating `bob` and learns the resource of `bob` at `i`. This makes *attack*(`google`, `bob`, `i`) hold, therefore the definition of conjuring retaliation scenario(Section 4.3) is met, that is *conjuring_retaliation_scenario*(`bob`, `i`, `google`) holds.

6 Tool Performance Outline

In the light of the various experiments conducted thus far, it seems fair to state that the performance of SATMC is satisfactory: the tool scales up to validating protocols under the MA threat model. In brief, SATMC translates the model checking problem

```
  ------------------------------------------------------------------------
  pop 1:

  GOALS: [secrecy_of_na]

  Step 0:         [step_1(a,c,ka,kc,dummy_nonce,dummy_nonce,set_72,1)]
  Step 1:         [overhear(c,a,a,c,crypt(pk(kc),pair(nonce(fn(na0,mr(a),1)),mr(a
))))]
  Step 2:         [decrypt_public_key_1(c,a,kc,1)]
  Step 3:         [impersonate_2(c,a,b,ka,kb,dummy_nonce,dummy_nonce,set_75,2,fn(
na0,mr(a),1))]
  Step 4:         [step_2(b,a,c,ka,kb,dummy_nonce,dummy_nonce,set_75,2,fn(na0,mr(
a),1))]
  Step 5:         [decrypt_public_key_1(b,a,kb,1)]
  ------------------------------------------------------------------------
```

Fig. 17 Partial order plan for `b`'s confidentiality attack on `na` in NSPK++

```
Attacks Found:                    true
Stop Condition Reached:           false
Formula statistics:
  Graph Construction Time:        0.905
  Graph Leveled Off:              no
  Generate clauses Time (sec):    0.282
  Clauses2SAT Time (sec):         0.422
  Sat_axioms(Init) Time (sec):    0.0
  Encoding Time (sec):            1.609
  Depth:                          6
  Atoms:                          2889
  Clauses:                        9145
Solving statistics:
  Total Solving Time (sec):       0.0
  Last Solving Time (sec):        0.0
Abstraction/Refinement statistics:
  Validation Time (sec):          0.0
  Models into POPs Time (sec):    0.312
  Refinement iterations:          0

Total Time:     1.921
```

Fig. 18 Runtimes for b's confidentiality attack on na in NSPK++

in a SAT formula and then invokes a SAT solver to determine whether a model for the formula exists [8]. In line with previous experiments based on the DY threat model, the total time spent by the SAT solver in order to analyse the formulae that the tool generates is always negligible with respect to the total time that the experiment requires. All experiments were conducted on an inexpensive machine, a 2.8GHz Intel Pentium 4 with 2GB RAM.

The experiments with confidentiality violations confirmed our expectations that the runtimes would not significantly differ from previous homologous experiments conducted under DY. In particular, the experiment for b's confidentiality attack upon nonce na ends in just a few seconds. This is plausible because the search space was never searched more deeply than at level 10. The average size of the

```
Attacks Found:                    true
Stop Condition Reached:           false
Formula statistics:
  Graph Construction Time:        17.315
  Graph Leveled Off:              12
  Generate clauses Time (sec):    66.235
  Clauses2SAT Time (sec):         37.344
  Sat_axioms(Init) Time (sec):    0.0
  Encoding Time (sec):            120.894
  Depth:                          14
  Atoms:                          34385
  Clauses:                        130385
Solving statistics:
  Total Solving Time (sec):       0.031
  Last Solving Time (sec):        0.031
Abstraction/Refinement statistics:
  Validation Time (sec):          0.0
  Models into POPs Time (sec):    14.047
  Refinement iterations:          0

Total Time:     134.972
```

Fig. 19 Runtimes for attack VI in NSPK++

SAT formula was of 9,000 atoms and 30,000 clauses. For example, Fig. 17 shows the raw partial order plan that expresses b's confidentiality attack on na in the classical protocol. Its polished version was given in Fig. 6.

The runtimes that the tool outputs along with this partial order plan are in Fig. 18. It is easy to read the number of atoms and clauses, the negligible SAT solving time, and the total time of less than 2 s.

The various attacks discussed about NSPK++, including retaliation and anticipation, required deepening the search space up to level 16. Because each level causes an exponential time growth, the total execution time increased up to approximately a few minutes. Also the size of the SAT formula increases considerably—for example up to approximately 35,000 atoms and 100,000 clauses for the retaliation experiments. This puts significant burden on the machine, because the size of the full search space is exponential in the number of atoms. Figure 19 shows the runtimes for attack VI in the classical protocol. This experiment generated the highest number of clauses among all our experiments: 130,385. The total runtime is however just above 2 min.

The early experiments on Google SSO did not terminate overnight. The tool would approximate depth 20 in the search space without providing an output. We thus focused on a simplified version of the Google SSO were the client is assumed to be able to check the content of the messages he received as any other principal. Rather than a naive browser accepting and redirecting everything, the client is an Enhanced Proxy Client processing and evaluating the messages he received before redirecting them. This soon led to good performance, so that the conjuring retaliation attack was found in 23 s for a trace of depth 24. The size of the SAT formula is indeed smaller than in the experiments on NSPK++, with approximately 3,500 atoms and 15,000 clauses. This can be appreciated from the runtimes in Fig. 20.

Moreover, SATMC also comes with a pretty-printing facility that expresses partial-order plans in a more user-friendly notation nearer to readable protocol traces. Figure 21 demonstrates the instance for the retaliation attack found on

```
Attacks Found:                  true
Stop Condition Reached:         false
Formula statistics:
  Graph Construction Time:      0.142
  Graph Leveled Off:            no
  Generate clauses Time (sec):  20.905
  Clauses2SAT Time (sec):       0.64
  Sat_axioms(Init) Time (sec):  0.0
  Encoding Time (sec):          21.687
  Depth:                        24
  Atoms:                        3622
  Clauses:                      15423
Solving statistics:
  Total Solving Time (sec):     0.0
  Last Solving Time (sec):      0.0
Abstraction/Refinement statistics:
  Validation Time (sec):        0.0
  Models into POPs Time (sec):  1.469
  Refinement iterations:        0

Total Time:     23.156
```

Fig. 20 Runtimes for Google retaliation attack

```
bob          o---(c2spA)-->*    i           : bob.i.uri
i            *---(sp2cA)-->o    bob         : idp.i.anyid.uri
bob          o--(c2idpA)-->*    idp         : idp.i.anyid.uri
idp          *--(idp2cA)-->*    bob         : i.{bob.idp}_inv(kidp).uri
bob          o---(c2spA)-->*    i           : i.{bob.idp}_inv(kidp).uri
i(bob)       o---(c2spB)-->*    google      : bob.google.cal
google       *---(sp2cB)~~>o    bob         : idp.google.id(google,0).cal
i(bob)       o---(c2spB)-->*    google      : google.{bob.idp}_inv(kidp).cal
google       *---(sp2cB)-->o    i(bob)      : resource(google,s(0))
google(bob)  o---(c2spC)-->*    i           : bob.i.mail
i            *---(sp2cC)~~>o    bob         : idp.i.id(i,s(s(0))).mail
i            *---(sp2cA)~~>o    bob         : resource(google,s(0))
google(bob)  o---(c2spC)-->*    i           : i.{bob.idp}_inv(kidp).mail
i            *---(sp2cC)-->o    google(bob) : resource(i,s(s(s(0))))
i            *~~~(sp2cA)-->o    bob         : resource(google,s(0))
i            *~~~(sp2cC)-->o    google(bob) : idp.i.id(i,s(s(0))).mail
```

Fig. 21 Google retaliation attack

Google SSO. The support that the pretty-printing facility offers to the human analyser, with such long attack traces especially, is remarkable.

7 Conclusions

This manuscript has shown how to validate classical and modern protocols under the Multi-Attacker threat model. Its contribution therefore is at least twofold, advancing both the protocol insights and the validation method.

In terms of protocol insights, various attacks have been found and the retaliation and anticipation scenarios have been defined and studied.

Retaliation teaches us that we can perhaps live with flawed protocols. We are used to go back to design when a protocol is found flawed, even if already deployed. However, an attack that can be retaliated may in practice convince an attacker to refrain from attacking in the first place. If the "cost" of attacking overdoes its "benefits" then the attacker will not be carried out. Retaliation makes that precondition hold.

Anticipation teaches us to ponder the entire sequence of events underlying an attack. An attack typically is an interleaving of legal and illegal steps rather than a single illegal action. Therefore, we may face a scenario, unreported so far, where a principal mounts an attack by successfully exploiting for his own sake the illegal activity initiated but not yet finalised by someone else. For example, this is routine for the present hackers' community.

As for the validation method, our work required a number of modifications and extensions. Concerning the modelling phase, the definition of the single attacker knowledge was generalised as any principal's knowledge, which also required a new fact to distinguish between private and non private knowledge. The validation phase yielded the innovative definitions of the retaliation and anticipation scenarios, and of the various attacks on the classical protocol. In particular, the retaliation scenarios went through various versions and refinements, with the final versions presented

here advancing on a recent publication [11]. The retaliation scenario can now be applied to the modern protocol.

On one hand, we were pleased to find out that no modifications were necessary to the state-of-the-art model checker SATMC, but its performance was seriously put at stake. The early, raw experiments on the classical protocol run up to 20 min but are currently reduced to just a few minutes. Performance was even more problematic with the modern deployed protocol because the early experiments running overnight did not return in the morning. Switching on a simplified variant of Google SSO allows us to perform our experiments and to discover the conjuring retaliation attack in only 23 s. So used as we are, as protocol analysers, to reasoning in terms of a DY attacker, we might think of our novel investigations as unrealistic. However, scenarios of retaliation and anticipation often appear in our era of computer networks, notably in the area of intrusion management. With the increasing awareness of computer security issues, successful attacks rarely are instantaneous moves without consequences. Rather, they often involve a combination of smaller illegal achievements or the breaking of subsequent protection levels to succeed. These are typically monitored by either administrators or other attackers, and hence attacks can be anticipated or retaliated. One of our findings is that current machine-assisted techniques of protocol analysis are adequate to tackling these scenarios. The field of protocol analysis seems bound to develop much further.

Acknowledgements Roberto Carbone offered enthusiastic and timely assistance with various points concerning the SATMC tool and abstract communication channels. Andrew Gordon's constructive criticism helped us detail the Multi-Attacker threat model. Luca Viganò continuously provided constructive feedback. Last, but not least, we would like to thank the anonymous reviewers for their invaluable comments, which helped us to improve this manuscript.

References

1. Abadi, M., Gordon, A.: A calculus for cryptographic protocols: the spi calculus. Inf. Comput. **148**(1), 1–70 (1999)
2. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: Proc. of the International Conference IFIP on Theoretical Computer Science (TCS'00), pp. 3–22. Springer, Heidelberg (2000)
3. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., Porth, C.: Bar fault tolerance for cooperative services. ACM SIGOPS Oper. Syst. Rev. **39**(5), 45–58 (2005)
4. Anderson, R.: Why cryptosystems fail. In: CCS93, pp. 217–227. ACMP (1993)
5. Armando, A., Basin, D.A., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV. Lecture Notes in Computer Science, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
6. Armando, A., Carbone, R., Compagna, L.: LTL model checking for security protocols. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF20), 6–8 July 2007, Venice, Italy. LNCS. Springer, Heidelberg (2007)
7. Armando, A., Carbone, R., Compagna, L., Cuellar, J., Abad, L.T.: Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In: Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008). ACM, New York (2008)
8. Armando, A., Compagna, L.: SATMC: a SAT-based model checker for security protocols. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04). LNAI, vol. 3229, pp. 730–733, Lisbon, Portugal. Springer, Heidelberg (2004)

9. Armando, A., Compagna, L.: SAT-based model-checking for security protocols analysis. Int. J. Inf. Secur. **6**(1), 3–32 (2007)
10. Arsac, W., Bella, G., Chantry, X., Compagna, L.: Attacking each other. In: Proc. of the 17th International Workshop on Security Protocols (CIWSP'09). Springer, Heidelberg (2009)
11. Arsac, W., Bella, G., Chantry, X., Compagna, L.: Validating security protocols under the general attacker. In: Proc. of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09). Springer, Heidelberg (2009)
12. AVISPA: AVISPA Library of security protocols. http://www.avispa-project.org/library/index.html
13. Backes, M., Pfitzmann, B.: Relating symbolic and cryptographic secrecy. In: IEEE Symposium on Security and Privacy (2005)
14. Bella, G.: Formal Correctness of Security Protocols. Information Security and Cryptography. Springer (2007)
15. Bella, G.: The rational attacker. http://www.dmi.unict.it/~giamp/Seminars/rationalattackerSAP08.pdf. Invited talk at SAP Research France, Sophia Antipolis (2008)
16. Bella, G.: What is correctness of security protocols? Springer J. Univers. Comput. Sci. **14**(12), 2083–2107 (2008)
17. Bella, G., Bistarelli, S.: Confidentiality levels and deliberate/indeliberate protocol attacks. In: Christianson, B., Crispo, B., Harbison, W.S., Roe, M. (eds.) Proc. of the 10th Security Protocols Workshop (SPW'02). LNCS 2845, pp. 104–119. SV (2004)
18. Bella, G., Bistarelli, S., Massacci, F.: Retaliation: can we live with flaws? In: Essaidi, M., Thomas, J. (eds.) Proc. of the Nato Advanced Research Workshop on Information Security Assurance and Security. Nato Through Science, vol. 6, pp. 3–14. IOS, Amsterdam (2006). http://www.iospress.nl/loadtop/load.php?isbn=9781586036782
19. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: Proceedings 27th Annual Symposium on the Theory of Computing, pp. 57–66. ACM (1995)
20. Blanchet, B.: Automatic verification of cryptographic protocols: a logic programming approach. In: Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27–29 August 2003, pp. 1–3. Uppsala, Sweden (2003)
21. Buttyán, L., Hubaux, J.-P., Čapkun, S.: A formal model of rational exchange and its application to the analysis of syverson's protocol. J. Comput. Secur. **12**(3,4), 551–587 (2004)
22. Caleiro, C., Viganò, L., Basin, D.: Metareasoning about security protocols using distributed temporal logic. In: Electronic Notes in Theoretical Computer Science (Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA 2004), vol. 125(1), pp. 67–89. http://www.sciencedirect.com (2005)
23. Caleiro, C., Viganò, L., Basin, D.: Relating strand spaces and distributed temporal logic for security protocol analysis. Log. J. IGPL **13**(6), 637–663 (2005)
24. Compagna, L.: SAT-based model-checking of security protocols. Phd, Università degli Studi di Genova, Italy, and University of Edinburgh, Scotland (2005). Available at www.ai-lab.it/compa/PhD-Thesis/main.ps
25. Dolev, D., and Yao, A.: On the security of public-key protocols. IEEE Trans. Inf. Theory **2**(29) 350–357 (1981)
26. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: proving security protocols correct. J. Comput. Secur. **7**, 191–230 (1999)
27. Gollmann, D.: On the verification of cryptographic protocols—a tale of two committees. In: Proc. of the Workshop on Secure Architectures and Information Flow, ENTCS 32. Elsevier Science (2000)
28. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Compiling and verifying security protocols. In: Parigot, M., Voronkov, A. (eds.) Proceedings of LPAR 2000. LNCS 1955, pp. 131–160. Springer, Heidelberg (2000)
29. Kremer, S., Raskin, J.-F.: Game analysis of abuse-free contract signing. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02), pp. 206–230. IEEE, New York (2002)
30. Lowe, G.: Breaking and fixing the needham-shroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) Proceedings of TACAS'96. LNCS 1055, pp. 147–166. Springer, Heidelberg (1996)
31. Lowe, G.: Towards a completeness result for model checking of security protocols. J. Comput. Secur. **7**(2–3), 89–146 (1999)
32. Maurer, U.M., Schmid, P.E.: A calculus for security bootstrapping in distributed systems. J. Comput. Secur. **4**(1), 55–80 (1996)

33. Needham, R.M.: Keynote address: the changing environment. In: Christianson, B., Crispo, B., Malcolm, J.A., Michael, R. (eds.) Proc. of the 7th Security Protocols Workshop (SPW'99). LNCS 1796, pp. 1–5. Springer, Heidelberg (2000)
34. Neuman, B.C., Ts'o, T.: Kerberos: an authentication service for computer networks, from IEEE communications magazine, september (1994). In: Stallings, W. (ed.) Practical Cryptography for Data Internetworks. IEEE, New York (1996)
35. OASIS. Security assertion markup language (SAML) v2.0. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security (2005)
36. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Comput. Secur. **6**, 85–128 (1998)
37. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions and composed keys is NP-complete. Theor. Comput. Sci. **299**, 451–475 (2003). http://www.loria.fr/∼rusi/pub/tcsprotocol.ps.gz
38. Ryan, P.Y.A., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A.W.: Modelling and Analysis of Security Protocols. AW (2001)