Verificação Automática dos Protocolos de Segurança Needham-Schroeder, WMF e CSA com a ferramenta Scyther*

Tadeu Jenuario[1,3], João Otávio Chervinski[1], Giulliano Paz[2,3], Rafael Beltran[2,3], Rafael Fernandes[1,3], Diego Kreutz[1,2,3]

[1]Laboratório de Estudos Avançados (LEA)
[2] Mestrado Profissional em Engenharia de Software (MPES)
[3] Universidade Federal do Pampa (UNIPAMPA)
{gnoario,joaootaviors,giulliano94,rbduarte96,faelsfernandes}@gmail.com

kreutz@acm.org

Abstract. *Security protocols (e.g., Needham-Schroeder, WMF, CSA, SSH, TLS) represent today the foundation of communications. One of the biggest challenges in designing such kind of protocols is to ensure that they are vulnerability free. Fortunately, there are different tools, such as Scyther, CryptoVerif, AVISPA and Tamarin Prover, that can be used for automated and formal verification of protocols. However, those tools are not widely known and, as a consequence, are rarely used in practice by protocol designers. This work starts to address this gap by presenting an introduction and a hands-on with Scyther, a tool designed for helping on the automated verification of security protocols, using three different protocols (Needham-Schroeder, WMF, and CSA).*

Resumo. *Protocolos de segurança (e.g., Needham-Schroeder, WMF, CSA, SSH, TLS) representam o alicerce das comunicações do mundo atual. Um dos desafios de projeto destes protocolos é garantir a sua própria segurança. Felizmente, existem ferramentas desenvolvidas especificamente para a verificação formal e automática de protocolos de segurança, como a Scyther, CryptoVerif, AVISPA e Tamarim Prover. Entretanto, estas ferramentas são ainda pouco conhecidas e utilizadas na prática por projetistas de protocolos e estudantes de computação. Este trabalho tem por objetivo contribuir para diminuir esta lacuna através da introdução e demonstração prática de utilização da ferramenta Scyther, desenvolvida para auxiliar na verificação automática de protocolos de segurança, utilizando três protocolos distintos (Needham-Schroeder, WMF e CSA).*

1. Introdução

Com a popularização da Internet, a segurança das comunicações tornou-se crítica para proteger os dados das entidades comunicantes. Atualmente, os mecanismos (e.g., geração e troca de chaves) e as propriedades essenciais de segurança da informação (e.g., confidencialidade, integridade e autenticidade) são asseguradas por conjuntos de protocolos específicos, como IKE, Needham-Schroeder, WMF, CSA, Kerberos, IPSec, SSH e TLS.

*Este artigo é uma versão estendida do paper [Jenuario et al. 2019], publicado no WRSeg 2019 (https://errc.sbc.org.br/2019/wrseg/) e selecionado entre os melhores trabalhos do evento para publicação em revista.

Um dos maiores desafios no desenvolvimento de protocolos de segurança é garantir a corretude do próprio protocolo, desde o projeto até a implementação. Devido a isso, a verificação automática e formal de protocolos, utilizando métodos formais e matemáticos, vem tornando-se cada vez mais frequente e imprescindível na comunidade de segurança [Chudnov et al. 2018, Li et al. 2018, Kreutz et al. 2019]. Pesquisas relatam, por exemplo, que o processo de verificação formal já contribuiu também para a correção de protocolos que estavam em utilização, mas vulneráveis, na comunidade por longos períodos de tempo [Dalal et al. 2010, Cremers et al. 2016, Affeldt and Marti 2013].

Existem diferentes ferramentas desenvolvidas especificamente para auxiliar o projeto e a verificação automática de protocolos, como Scyther [Cremers 2006], AVISPA [Armando, A., et al. 2005], ProVerif [Blanchet et al. 2018] e Tamarim Prover[1] [Meier et al. 2013]. No entanto, estas ferramentas ainda são pouco conhecidas e utilizadas pela comunidade. Por exemplo, analisando alguns dos principais cursos de computação do Rio Grande do Sul e as últimas três edições do WRSeg e do SBSeg, dois dos principais eventos de segurança do Brasil, não foram encontrados trabalhos sobre verificação formal de protocolos de segurança utilizando essas ferramentas.

O objetivo deste trabalho é contribuir para a difusão do conhecimento e da importância de utilizar ferramentas de verificação automática e formal de protocolos de segurança. Neste sentido, as principais contribuições do trabalho podem ser resumidas em: (a) introdução às semânticas operacionais básicas da ferramenta Scyther; (b) descrição didática do protocolo Needham-Schroeder de chaves assimétricas [Needham and Schroeder 1978] utilizando a semântica da ferramenta; (c) apresentação e descrição dos protocolos WMF (troca de chaves simétricas com agente confiável) e CSA (Cifra Simétrica e Assimétrica) na semântica da Scyther; e (d) demonstração e discussão do processo de análise e correção de falhas dos protocolos.

2. A Ferramenta Scyther: Semânticas Operacionais

Assim como ocorre com as demais ferramentas, a verificação automática de protocolos com a Scyther requer a utilização de uma linguagem própria [Cremers 2006]. A seguir são apresentadas algumas das principais semânticas operacionais da ferramenta, que servirão como base para o projeto e análise formal dos protocolos de Needham-Schroeder, WMF e CSA (ver Seções 3 e 6).

Termos atômicos: a ferramenta Scyther manipula termos de modo que o protocolo apresente o comportamento desejado. Os termos atômicos, ou tipos específicos de dados, servem para definir os dados que serão utilizados no protocolo. Entre eles estão: o *var*, que define variáveis para armazenar os dados recebidos de uma comunicação; o *const*, que define constantes que são geradas para cada instanciação de uma função e são consideradas variáveis locais; e o *fresh*, que representa variáveis que serão iniciadas com valores pseudo-aleatórios.

Chaves assimétricas: a estrutura de chave públicas ou assimétricas é predefinida pela ferramenta, sendo $sk(X)$ correspondente a uma chave privada de X e $pk(X)$ a chave pública correspondente. Um dado ni cifrado utilizando uma chave pública (e.g., $\{ni\}pk(X)$) só poderá ser decifrado com a chave privada correspondente ($sk(X)$).

[1] `https://tamarin-prover.github.io/`

Tipos predefinidos: determinam o comportamento de uma função ou termo. Por exemplo, o tipo **Agent** é utilizado para criar um agente que irá interagir nas comunicações. **Function** é um tipo especial que define um termo como sendo uma função que pode receber uma lista de termos como parâmetro. O tipo **Nonce** é considerado o tipo padrão para termos e é destinado ao armazenamento de valores utilizados durante a troca de mensagens.

Tipos básicos de eventos, como **send** (enviar) e **recv** (receber), são utilizados para a comunicação entre os agentes de um protocolo (e.g., Alice e Bob). É comum que cada evento de envio possua um evento de recebimento correspondente (e.g., **send_1** e **recv_1**). Para enviar uma mensagem de Alice para Bob, contendo o dado ni cifrado com a chave pública de Bob, basta utilizar a sintaxe $send\_1(Alice, Bob, \{ni\}pk(Bob))$. Para receber a mensagem enviada de Alice para Bob, basta utilizar a sintaxe $recv\_1(Alice, Bob, \{ni\}pk(Bob))$. Bob irá receber a mensagem de Alice e utilizar a sua chave privada correspondente para decifrar o dado contido na variável ni.

Eventos de afirmação (*claim*) são utilizados em especificações de funções para modelar propriedades de segurança. Na prática, após afirmar que uma variável é secreta, a ferramenta irá verificar se a afirmação procede durante a execução do protocolo, i.e., verificar se somente as partes comunicantes possuem acesso ao dado secreto. Por exemplo, para afirmar que uma variável ni é secreta, utiliza-se o termo **Secret** dentro de um evento de afirmação, i.e., $claim(Bob, Secret, ni)$. Também pode-se utilizar o termo **Nisynch** dentro de um evento de afirmação, i.e., $claim(Bob, Nisynch)$ para verificar se todas as mensagens recebidas pelo receptor foram de fato enviadas pelo parceiro de comunicação e não por um agente desconhecido.

3. O Protocolo Needham-Schroeder

O protocolo Needham-Schroeder fornece um método de autenticação para dois participantes em uma rede insegura utilizando criptografia assimétrica [Needham and Schroeder 1978]. Entretanto, o protocolo original apresenta falhas de segurança [Lowe 1995]. O objetivo é demonstrar e corrigir as falhas de segurança do protocolo utilizando a ferramenta Scyther.

A Figura 1 apresenta um diagrama da comunicação entre Alice e Bob utilizando o protocolo Needham-Schroeder. É assumido que cada usuário possui a chave privada e a chave pública do seu par. Alice deseja comunicar-se com Bob e, para tanto, requisita ao servidor de chaves a chave pública do Bob. Na sequência, Alice gera e envia para Bob um *nonce* ni juntamente com sua identificação. A mensagem é cifrada com a chave pública do Bob. Ao receber a mensagem, Bob decodifica-a e recupera o *nonce* ni e a identificação da remetente. Bob lê a identificação de Alice e requisita a respectiva chave pública para o servidor de chaves. No próximo passo, Bob gera e envia para Alice um novo *nonce* nr, cifrado com a chave pública da receptora. Ao receber a mensagem, Alice verifica que o *nonce* ni recebido é o mesmo que foi enviado para Bob anteriormente. Finalmente, Alice então envia para Bob o *nonce* nr. Ao receber e verificar os respectivos *nonces*, utilizando as respectivas chaves privadas, Alice e Bob são capazes de confirmar a autenticidade das mensagens.

O Algoritmo 1 apresenta o protocolo Needham-Schroeder na semântica da Scyther. As chaves públicas pk e privadas sk são declaradas globalmente (linhas 1 e

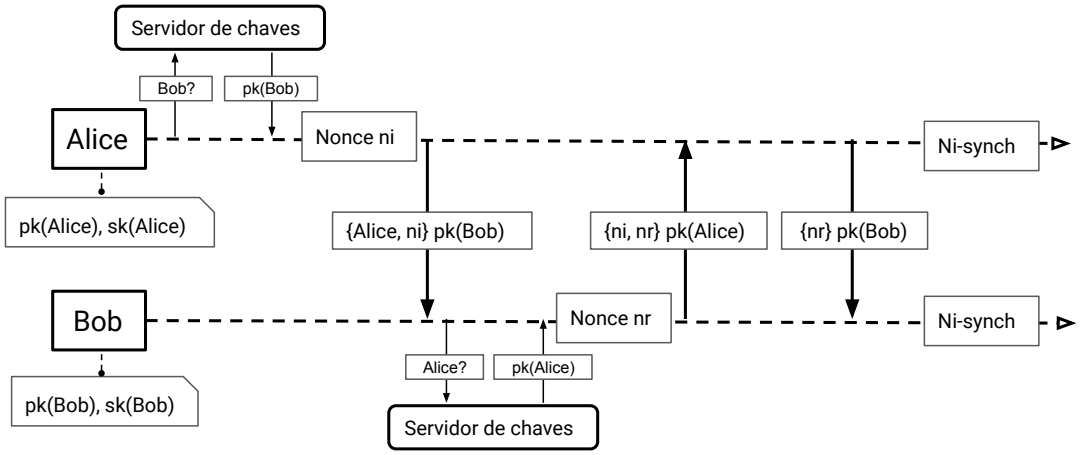


Figura 1. Alice e Bob utilizando o protocolo Needham-Schroeder

2), embora sejam atribuídas autonomamente a Alice e Bob pela ferramenta. O comando *inversekeys* (linha 3) determina que as chaves pk e sk são inversas, ou seja, ao cifrar com a chave pública só é possível decifrar com a chave secreta e vice-versa. Para a execução de ataques ao protocolo, é definido um agente não-confiável *Eve* (linhas 5 e 6).

Algoritmo 1: Algoritmo Needham-Schroeder na semântica da Scyther

```
1  const pk: Function;
2  secret sk: Function;
3  inversekeys (pk,sk);
4  const Eve: Agent;
5  untrusted Eve;
6  protocol NS(Alice,Bob,Eve){
7     role Alice{
8        fresh ni: Nonce;
9        var nr: Nonce;
10       send_1(Alice,Bob,{Alice,ni}pk(Bob));
11       recv_2(Bob,Alice,{ni,nr}pk(Alice));
12       send_3(Alice,Bob,{nr}pk(Bob));
13       claim(Alice,Secret,ni);
14       claim(Alice,Secret,nr);
15       claim(Alice,Nisynch);
16    }
17    role Bob{
18       var ni: Nonce;
19       fresh nr: Nonce;
20       recv_1(Alice,Bob,{Alice,ni}pk(Bob));
21       send_2(Bob,Alice,{ni,nr}pk(Alice));
22       recv_3(Alice,Bob,{nr}pk(Bob));
23       claim(Bob,Secret,ni);
24       claim(Bob,Secret,nr);
25       claim(Bob,Nisynch);
26    }
27 }
```

A chamada à função **protocol** (linha 6) marca o início da especificação do protocolo NS, com três agentes (Alice, Bob e Eve), sendo que Alice e Bob possuem papéis (**role**) explícitos. A variável ni (linha 8), do tipo **Nonce** e antecedida por **fresh**, irá conter um valor pseudo-aleatório (e.g., linha 10). Já a variável nr (linha 9), do mesmo tipo, mas **var** ao invés de **fresh**, é utilizada para armazenar valores recebidos (e.g., linha 11).

Cada evento de envio (**send_1**) possui um evento de recebimento (**recv_1**) correspondente (e.g., linhas 10 e 20). A sintaxe do evento **send_1** (linha 10) indica que a transmissão é de Alice para Bob, os dados enviados, cifrados com a chave pública do Bob $pk(Bob)$, são "*Alice*" e ni. No agente Bob (linha 20), há um evento com sintaxe idêntica, cuja única diferença é o tipo, i.e., **recv** ao invés de **send**. Na sequência, Bob envia os valores ni e nr para Alice (linha 21). Alice recebe a resposta (linha 11), verificar o valor do ni e envia nr para Bob (linha 12), que recebe e verifica o valor (linha 22).

Finalmente, as afirmações **claim** (linhas 13, 14, 15 23, 24, 25) definem os requisitos de segurança do protocolo. No caso, as afirmações criadas verificam se os *nonces* gerados por Alice (ni) e Bob (nr) permanecem secretos durante as comunicações (**claim Secret**) e se as mensagens são de fato trocadas entre eles apenas (**claim Nisynch**).

4. O protocolo WMF

O protocolo WMF (*Wide Mouth Frog*) realiza a troca da chave simétrica, compartilhada entre Alice e Bob, através de um agente confiável (Charles) utilizando uma chave compartilhada denominada *private key* [Kelsey et al. 1997, Velibor et al. 2016]. Esta chave é utilizada exclusivamente para distribuição das chaves. Resumidamente, no WMF, quem gera a chave de sessão, utilizada por Alice e Bob, é o agente que inicia a comunicação (e.g., Alice).

A Figura 2 ilustra o funcionamento do WMF. Para trocar a chave de sessão, Alice envia uma mensagem ao agente confiável Charles contendo sua identificação, um *timestamp* Ti, a identificação do destinatário (Bob) e a nova chave de sessão. Charles recebe e envia uma mensagem a Bob contendo sua identificação, um *timestamp* Ts, o identificador do remetente (Alice) e a chave de sessão compartilhada, criada e enviada por Alice.

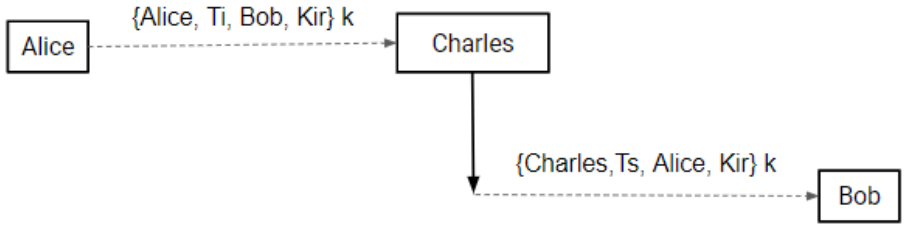


Figura 2. Alice e Bob utilizando o protocolo WMF

O Algoritmo 2 descreve o protocolo WMF na semântica da Scyther. Na linha 1 é declarado um novo tipo de termo, denominado *timestamp*. O protocolo WMF é iniciado na linha 6, tendo como agentes Alice, Bob e Charles.

Algoritmo 2: Algoritmo WMF na semântica da Scyther

```
1  usertype Sessionkey;
2  usertype TimeStamp;
3  protocol WMF(Alice,Bob,Charles){
4     role Alice{
5        fresh Kir: Sessionkey;
6        fresh Ti: TimeStamp;
7        send_1( Alice,Charles,{Alice,Ti,Bob,
       Kir}K(Alice,Charles));
8        claim_ Alice1(Alice,Secret,Kir);
9     role Bob{
10       var Ts: TimeStamp;
11       var Kir: Sessionkey;
12       recv_2(Charles,Bob,{Charles,Ts, Alice,
       Kir}K(Bob,Charles));
13       claim_Bob1(Bob,Secret,Kir);
14       claim_Bob2(Bob,Nisynch);
15    role Charles{
16       var Kir: Sessionkey;
17       fresh Ts: TimeStamp;
18       var Ti: TimeStamp;
19       recv_1( Alice,Charles, Alice,{ Alice,
            Ti, Bob,Kir}K( Alice,Charles));
20       send_2(Charles,Bob,{Charles,Ts,
       Alice, Kir}K(Bob,Charles));
21    }
22 }
```

No papel (**role**) Alice (linha 4) são criadas e inicializadas com valores pseudo-aleatórios as variáveis *Kir* (chave de sessão) e *Ti* (linhas 5 e 6). Na linha 7, Alice envia

para Charles sua identificação, um *timestamp*, a identificação do recebedor e a chave de sessão. A verificação da chave de sessão é realizada na afirmação (**claim**) da linha 8.

No papel (**role**) Bob (linha 9) são criadas as variáveis *Ts* e *Kir* (linhas 10 e 11), que irão armazenar o *timestamp* e a chave de sessão recebidos de Charles, respectivamente. Bob recebe a mensagem de Charles (linha 12) contendo a identificação de Charles, o *timestamp* de envio de Charles, a identificação de Alice e a chave de sessão. A verificação da chave de sessão e da segurança da comunicação é realizada nas afirmações das linhas 13 e 14.

No papel do agente confiável Charles (**role** definido na linha 15) são criadas as variável *Kir*, que irá receber a chave de sessão de Alice (linha 16), um *timestamp Ts* (linha 17) e **Ti**, que irá armazenar o *timestamp* enviado por Alice (linha 18). Charles recebe a mensagem de Alice contendo o identificador dela, o identificador do destinatário (Bob) e a chave de sessão (linha 19). A chave de sessão é encaminhada por Charles para Bob, incluindo um novo *timestamp Ts* (linha 20).

5. O Protocolo CSA

Para ilustrar a utilização da Scyther em protocolos que utilizam simultaneamente criptografia simétrica e assimétrica, foi criado um protocolo didático denominado CSA (Criptogria Simétrica e Assimétrica). O protocolo é utilizado por dois agentes quaisquer (e.g., Alice e Bob) para trocar dados (e.g., uma imagem) de forma segura. Uma chave simétrica secreta, compartilhada entre ambos os agentes, é utilizada para cifrar os dados. Já as respectivas chaves privada/pública são utilizadas para assinar e verificar a autenticidade da mensagem, garantindo também o não-repúdio (i.e., Alice não pode negar que enviou uma mensagem assinada com sua chave privada e vice-versa). De forma similar a protocolos como o SSH e o TLS, as chaves privada/pública do CSA podem ser também utilizadas para gerar ou atualizar a chave simétrica secreta compartilhada entre Alice e Bob.

A Figura 3 ilustra um diagrama de comunicação entre Alice e Bob utilizando o CSA. É assumido que cada agente possui o seu respectivo par de chaves privada/pública e uma chave secreta compartilhada entre ambos. No exemplo, Alice envia uma imagem ($NonceImagem$) juntamente com a identificação de quem deve receber a imagem (Bob). A mensagem é cifrada com a chave secreta simétrica $sKey$ e assinada com a chave secreta sk do remetente, no caso Alice. Ao receber a imagem, Bob solicita ao servidor de chaves a chave pública de Alice para validar a origem da mensagem, isto é, verificar sua autenticidade.

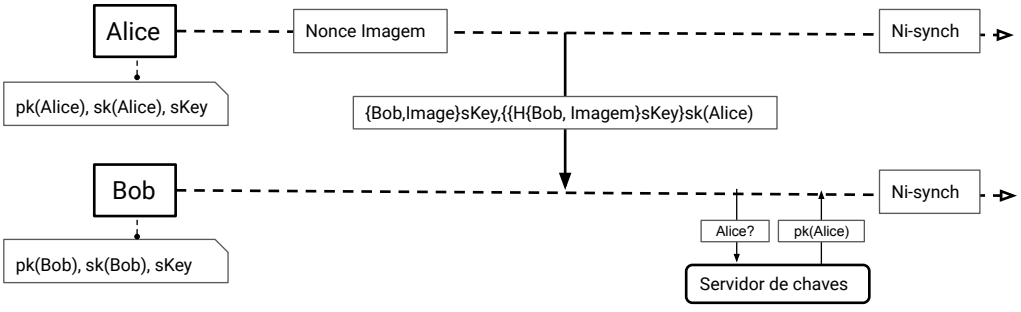


Figura 3. Alice e Bob utilizando o protocolo CSA

O Algoritmo 3 descreve o protocolo CSA na semântica da ferramenta Scyther. As chaves públicas pk e privadas sk são declaradas globalmente (linhas 1 e 2) e utilizadas para gerar e verificar as assinaturas das mensagens. O comando *inversekeys* (linha 3) determina que as chaves pk e sk são inversas, o que é o padrão para chaves públicas, utilizadas em criptografia assimétrica, na Scyther. A chave secreta $sKey$ é declarada também globalmente (linha 4), representando a chave simétrica compartilhada entre Alice e Bob. Na sequência (linha 5), é declarada a função H (**hashfunction**, na linha 5), que é utilizada para a geração da hash (ou *digest*) das mensagens. A assinatura da mensagem (dentro da função de envio na linha 9) é representada pela cifra do *digest* da mensagem utilizando a chave privada de Alice.

Algoritmo 3: Algoritmo Simétrico e Assimétrico na semântica da Scyther

```
1  const pk: Function;
2  secret sk: Function;
3  inversekeys (pk,sk);
4  secret sKey: Function;
5  hashfunction H;
6  protocol CSA(Alice,Bob){
7     role Alice{
8        fresh Imagem: Nonce;
9        send_1(Alice,Bob,{Bob,Imagem}sKey,
           {H({Bob,Imagem}sKey)}sk(Alice));
10       claim(Bob,Secret,Imagem);
11       claim(Alice,Nisynch);
12    }
13    role Bob{
14       var Imagem: Nonce;
15       recv_1(Alice,Bob,{Bob,Imagem}sKey,
           {H({Bob,Imagem}sKey)}sk(Alice));
16       claim(Bob,Secret,Imagem);
17       claim(Bob,Nisynch);
18    }
19 }
```

No protocolo CSA (linha 6) são definidos dois agentes, Alice e Bob. No papel (**role**) Alice (linha 7) é criada a variável $Imagem$ (linha 8) do tipo **Nonce**, antecedida por **fresh**, o que significa que a variável será automaticamente inicializadas com um valor pseudo-aleatório, que representará os dados da imagem. Alice envia a imagem para Bob (linha 9). A mensagem contém o identificador do Bob e a imagem cifrados ($\{Bob, Imagem\}sKey$) utilizando a chave simétrica $sKey$ e a assinatura da mensagem utilizando a chave privada sk de Alice ($\{H(\{Bob, Imagem\}sKey)\}sk(Alice)$).

Na papel (**role**) Bob é criada a variável $Imagem$ do tipo **Nonce** (linha 15), antecedida por **var**, para armazenar a imagem recebida (linha 16). Bob recebe a mensagem, verifica a assinatura utilizando a chave pública da Alice e decifra a imagem utilizando a chave compartilhada $sKey$ (linha 15).

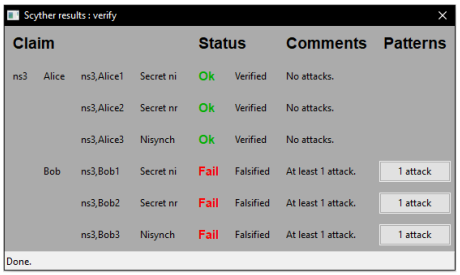
Para comprovar a confidencialidade das mensagens trocadas, são definidos os eventos de afirmação nas linhas 10, 11, 16 e 17. Estas **claims** verificam a integridade da $Imagem$ e a autenticidade das mensagens (**claim Nisynch**).

6. Análise dos Protocolos com a Ferramenta Scyther

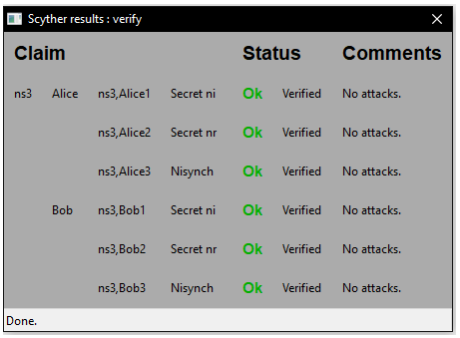
6.1. Análise do protocolo Needham-Schroeder

Ao analisar o código do Algoritmo 1 na ferramenta Scyther, é gerado um relatório que aponta a existência de falhas no protocolo, como pode ser visto na Figura 4a. Na coluna **Claim** são apresentadas quatro informações, o protocolo testado (*NS*), os agentes analisados (Alice e Bob), um par de valores que servem como um identificador único para cada evento (e.g., $ns, Alice1$) e o evento de afirmação (e.g., $Secret\ ni$). Nas colunas **Status**,

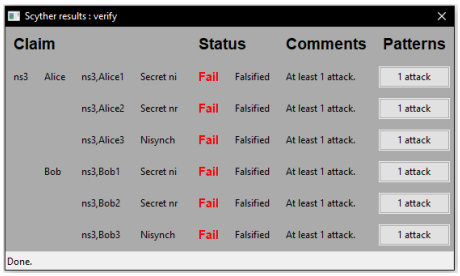
Comments e **Patterns** são reportados os eventuais ataques que o protocolo é suscetível. Como pode ser observado, há pelo menos um ataque no protocolo Needham-Schroeder, que afeta os eventos $ns, Bob1$ e $ns, Bob2$. Nestes casos, a ferramenta gera também um grafo com os passos de execução do(s) ataque(s), como ilustrado na Figura 5.



(a) Verificação do protocolo Needham-Schroeder



(b) Verificação do protocolo corrigido



(c) Comprometimento da chave secreta

Figura 4. Resultados gerados pela Scyther

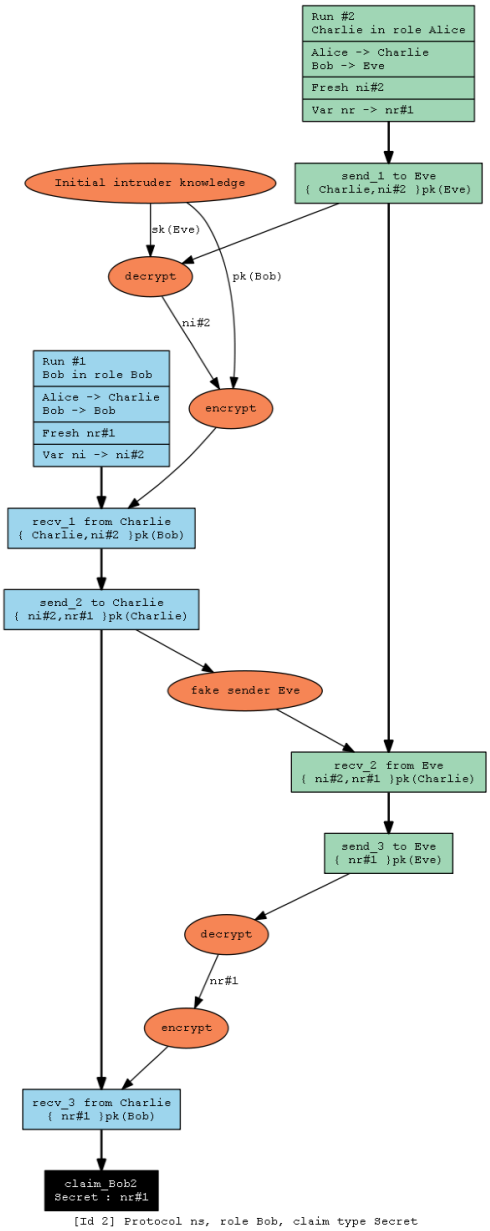


Figura 5. Diagrama do ataque ao protocolo

Para comprometer a comunicação entre Alice e Bob, Eve (agente malicioso) precisa apenas convencer Alice de que é Bob. Alice envia a Eve uma mensagem $\{Alice, ni\}$ cifrada com a chave pública $pkEve$. Como Eve possui a chave privada correspondente, ele consegue ler o conteúdo da mensagem. Eve então cifra e envia a mensagem para Bob. Bob, sem desconfiar, pois recebeu uma mensagem cifrada com sua chave pública, responde ao atacante com a mensagem $\{ni, nr\}pkAlice$. O agente malicioso simplesmente

encaminha a mensagem para Alice. Para confirmar o recebimento do $nonce\ nr$, Alice responde ao atacante com $\{nr\}pkEve$. O atacante decifra a mensagem, descobre o valor de nr, cifra e envia a mensagem para Bob. Bob confirma o valor de nr (autenticação) e acredita que está comunicando-se com Alice. A partir deste ponto, todas as mensagens trocadas entre Alice e Bob passam primeiro pelo atacante. Observem que o ataque é simples. Eve precisa apenas estar entre Alice e Bob e conhecer a chave pública de Bob.

Para corrigir esta falha do protocolo, Bob deve adicionar a sua identidade na resposta à primeira mensagem de Alice (ni, nr, Bob na linha 20 do Algoritmo 1). Com isto, Alice consegue descobrir que a identidade contida na mensagem é diferente da identidade de Eve, para quem está enviando as suas mensagens. Neste caso, Alice simplesmente encerra a troca de mensagens. Com o algoritmo corrigido (linha 20), o resultado da análise da Scyther pode ser visto na Figura 4b. Como pode ser observado, um simples detalhe de especificação pode comprometer toda a segurança do protocolo. Isto demonstra o quão importante é a utilização de ferramentas de verificação automática de protocolos.

Para observar o que ocorre no protocolo, é possível definir um agente não-confiável Eve ("$compromised\ sk(Eve)$;"), capaz de comprometer as chaves privadas dos demais agentes. Tomando como exemplo o Algoritmo 1, a ferramenta considera a existência de um agente malicioso que conhece as chaves secretas de Alice e Bob. Isto permite que o atacante comprometa as comunicações, conforme a Figura 4c.

6.2. Análise do protocolo WMF

O WMF apresenta uma falha, como pode ser observado na Figura 6a. A falha é detalhada na Figura 6b, que ilustra o diagrama do ataque. A falha acontece na mensagem de Charles para Bob, uma vez que é enviado apenas o *timestamp* do Charles. A solução mais simples é incluir também o *timestamp* da Alice na mensagem enviada para Bob. O resultado desta solução pode ser visualizado na Figura 6c.

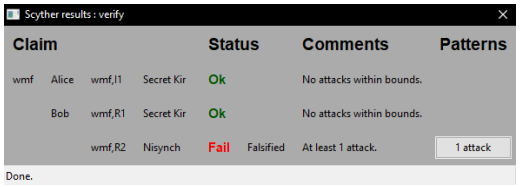
6.3. Análise do protocolo CSA

Para o caso do código do Algoritmo 3, como pode ser observado na Figura 7a, a comunicação entre `Alice` e `Bob` é segura segundo a análise automática da ferramenta. Em outras palavras, não há nenhum indicativo de falha no *Status* do relatório, isto é, tudo está como *Ok*, verificado e confirmado como sem ataques.

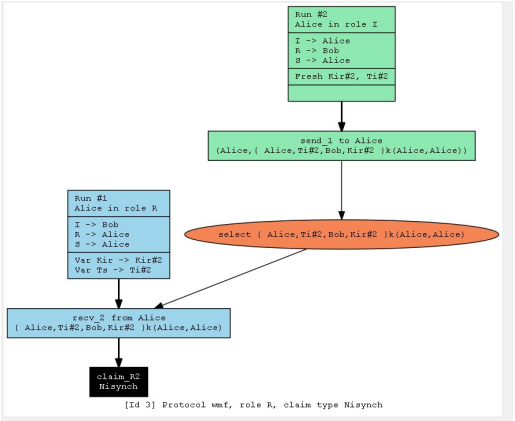
Para observar o comportamento do protocolo, pode-se definir propositalmente um agente não-confiável Eve, que é capaz de comprometer as chaves utilizadas na comunicação entre Alice e Bob. Tomando como exemplo o Algoritmo 3, a ferramenta considera a existência de um agente malicioso que conhece as chaves privadas (secretas) de Alice e Bob. Entretanto, apesar de Eve poder comprometer a autenticidade das mensagens (análise da Figura 7b), o atacante não consegue comprometer a confidencialidade dos dados (i.e., imagem) sem conhecer também a chave secreta compartilhada $sKey$.

7. Conclusão

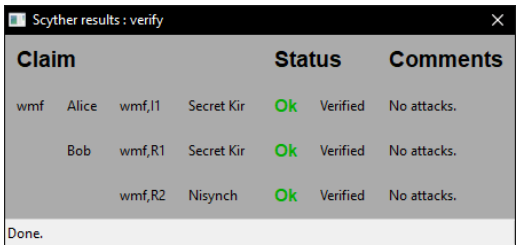
Este artigo apresentou uma introdução à ferramenta de verificação automática de protocolos Scyther, utilizando como exemplos os protocolos Needham-Schroeder (criptografia assimétrica), WMF (criptografia simétrica) e CSA (criptografia simétrica e assimétrica). Como pode ser visto nas análises da Seção 6, os protocolos Needham-Schroeder e WMF



(a) Verificação do protocolo WMF

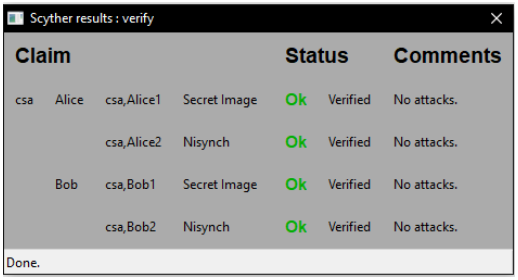


(b) Diagrama do ataque ao WMF

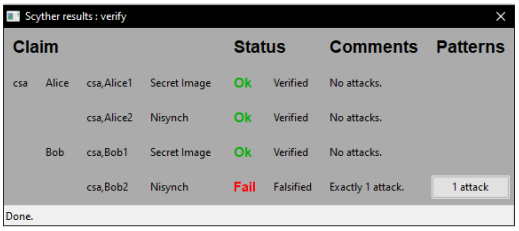


(c) Correção do WMF

Figura 6. Protocolo WMF



(a) Verificação do algoritmo 3 (CSA)



(b) Chave comprometida (CSA)

Figura 7. Protocolo CSA

possuem falhas de segurança graves, que podem ser rápida e facilmente detectadas e corrigidas com a utilização da Scyther. No caso de protocolos como o CSA, é possível observar o que acontece quando há um agente malicioso que compromete um subconjunto de chaves secretas utilizadas na comunicação entre Alice e Bob, por exemplo. Em síntese, estes exemplos práticos, reais, destacam a importância da disseminação do conhecimento e da utilização de ferramentas de verificação formal de protocolos.

Referências

Affeldt, R. and Marti, N. (2013). Towards Formal Verification of TLS Network Packet Processing Written in C. In *7th PLPV*, pages 35–46. ACM.

Armando, A., et al. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer.

Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. (2018). ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. `https://bit.ly/2OUlH7f`.

Chudnov, A., Collins, N., Cook, B., Dodds, J., Huffman, B., MacCárthaigh, C., Magill, S., Mertens, E., Mullen, E., Tasiran, S., Tomb, A., and Westbrook, E. (2018). Continuous Formal Verification of Amazon s2n. In *Computer Aided Verification*, pages 430–446.

Cremers, C., Horvat, M., Scott, S., and v. d. Merwe, T. (2016). Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *IEEE SP*.

Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven.

Dalal, N., Shah, J., Hisaria, K., and Jinwala, D. (2010). A comparative analysis of tools for verification of security protocols. *Int. J. of Comm., Network and System Sciences*, 3(10):779.

Jenuario, T., Chervinski, J. O., Paz, G., and Kreutz, D. (2019). Verificação Automática de Protocolos de Segurança com a ferramenta Scyther. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*, Alegrete-RS, Brasil. `http://errc.sbc.org.br/2019/wrseg/papers/jenuario2019verificao.pdf`.

Kelsey, J., Schneier, B., and Wagner, D. (1997). Protocol interactions and the chosen protocol attack. In *International Workshop on Security Protocols*, pages 91–104. Springer.

Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Trans. Priv. Secur.*, 22(2):8:1–8:36.

Li, L., Sun, J., Liu, Y., Sun, M., and Dong, J. (2018). A Formal Specification and Verification Framework for Timed Security Protocols. *IEEE Trans. on Soft. Engineering*, 44(8):725–746.

Lowe, G. (1995). An Attack on the Needham- Schroeder Public- Key Authentication Protocol. *Information processing letters*, 56(3).

Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer.

Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

Velibor, S., Ivana, O., and Ramo, S. (2016). Comparative analysis of some cryptographic systems. Technical Report 15, BUSINESS INFORMATION SECURITY CONFERENCE.