Verifying Group Authentication Protocols by Scyther

Huihui Yang, Vladimir Oleshchuk*, and Andreas Prinz
University of Agder
Kristiansand, Norway
{huihui.yang, vladimir.oleshchuk, andreas.prinz}@uia.no

Abstract

Scyther [1] is a tool designed to formally analyze security protocols, their security requirements and potential vulnerabilities. It is designed under the perfect or unbreakable encryption assumption [2], which means that an adversary learns nothing from an encrypted message unless he knows the decryption key. To our best knowledge, most protocols analyzed using Scyther are widely used standards and their complexity are limited. In this paper, we use Scyther to analyze two complex group authentication protocols [3] and their security properties. Due to the design goals and limitations of Scyther, we have only checked a subset of the security properties, which show that the group authentication protocols provide mutual authentication, implicit key authentication and they are secure against impersonation attack and passive adversaries. To achieve this, we have extended the expressing ability of Scyther based on some reasonable assumptions.

Keywords: formal verification, group authentication, Scyther

1 Introduction

There are two main approaches to the verification of security of protocols: provable security [4–6] and formal methods [7, 8]. Scyther [1] is one of the formal verification tools and is designed for the automatic verification of security protocols. The adversary model of Scyther is predefined, which is Dolev-Yao' model [9]. This approach has simplified the formalization of security protocols and makes it easier to start to work with Scyther for new users. Compared with other formal verification tools, such as SPIN [10, 11] (language Promela), the specification language of Scyther is no complicated and thus fast to learn. Scyther also outperformed some other state-of-the-art protocol verification tools, for instance, ProVerif tool [12]. In addition, Scyther can provide classes of protocol behavior compared with just single attack traces provided by other tools [13]. As for as we know, Scyther has already been used to verify different types of protocols, including authentication protocols (e.g., IKEv1 [14] and IKEv2 [14] protocol suites and the ISO/IEC 9798 [15, 16] family) and authenticated key exchange (AKE) protocols [17] (e.g., HMQV [18] , NAXOS [19]). The common for all these protocols is that their complexity is limited. In [20], we tried to use Scyther to analyze two complicated group authentication protocols, originally proposed in [3]. The main purpose of these group authentication protocols is to improve authentication efficiency for large groups. The relation between the authenticator and users to be authenticated is one to one. However, in this type group authentication protocols, the authenticator can authenticate multiple users at the same time. If the group authentication protocol proceeds successfully, mutual authentication should be achieved and a group session key will be agreed on.

In [3], a general framework was proposed for two types group authentication protocols by implementing different cryptographic primitives. The main difference between protocols of Type I and Type II is that an authenticator in Type II has PKI-based certificate while authenticators in Type I does not,

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 7:2 (June 2016), pp. 3-19
*Corresponding author: Jon Lilletunsvei 9, Grimstad, Norway, Tel: +47-37233212, Web: http://www.uia.no/kk/profil/vladimao

but all protocols constructed under this general framework were claimed in [3] to satisfy several security requirements, including security against passive adversaries, against impersonation attacks, providing mutual authentication, implicit authentication, forward and backward secrecy. To demonstrate how to construct protocols of both types, two examples were provided in [3]: one is based on the discrete logarithm problem (DLP) [21], and the second one is based on the elliptic curve discrete logarithm problem (ECDLP) [22] respectively. However, only protocols of both types based on DLP are analyzed, because ECDLP can be seen as a special type of DLP. Once the DLP-based protocols are proven to satisfy those security requirements, it can be concluded that ECDLP-based protocols will satisfy the same security requirements because of the way how to formalize DLP and ECDLP in Scyther.

This paper is an extension of [20]. The following two innovations are added. First of all, in [20], we only discussed the case where the group size was three, while in this paper, we analyze the cases for groups containing two, three and four members. More importantly, we show how to formalize DLP-based protocols of both Type I and Type II when the number of group members is N ($N \geq 3$). Secondly, we analyze some new properties of the protocols, including “Alive” and “Nisynch”. More details will be given in Sections 3, 4 and 5.

The rest of this paper is organized as follows. In the next section, we describe the general framework and the DLP-based protocols that we analyze in this paper. Then we introduce the model checking tool Scyther, its adversary model and specification language in Section 3. In Sections 4 and 5, we describe the details of how to use Scyther the formalize the DLP-based protocols of both Type I and Type II, including modeling difficult mathematical problems, security requirements and the algorithms that formalize these protocols when the number of group users vary. Finally, we conclude this paper in the last section.

2 Description of the Group Authentication Protocols

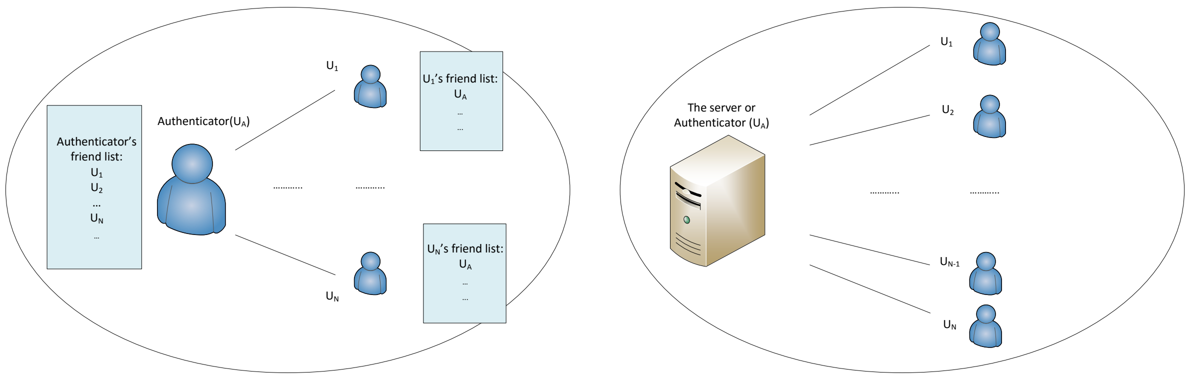
In this section, we describe the group authentication protocols. In Subsection 2.1, we briefly explain when and where these group authentication protocols can be applied and their main purposes. Next the message flow within the general framework and details about the DLP-based protocols will be explained in Subsections 2.2 and 2.3 respectively.

2.1 Usage scenarios

Two usage scenarios are considered in [3] and corresponding group authentication protocols are proposed (Type I and Type II). The main difference between proposed protocols is that the authenticator in Type II has a certificate but the authenticator in Type I does not. As shown in Fig. 1(a), the authenticator has a friend list, but members in this list may or may not know each other. Every time before group meeting, the authenticator first selects group members and then he needs to authenticate every member in this group. Since all members have already registered as the authenticator’s friends, we assume they share some secrets with the authenticator before the authentication. In Type II (Fig. 1(b)), the authenticator is a server and needs to authenticate a couple of users not necessary known in advance. In this case, the server should possess a certificate to perform the group authentication.

2.2 The general framework

Assume there are N members in user group $\mathbb{U}$. The message flow of the general framework proposed in [3] can be described in the following four steps.



(a) Usage scenario 1: Type I (b) Usage scenario 2: Type II

Figure 1: Two types of usage scenarios

1) $U_A \rightarrow U_1 : ID_A, UID, X, C_0, MAC_A$.
2) $U_i \rightarrow U_{i+1} : ID_i, UID, X, KP_U, C_i, MAC_i$, where $1 \leq i \leq N-1$.
3) $U_N \rightarrow U_A : ID_N, KP_U, C_N, MAC_N$.
4) $U_A \rightarrow \mathbb{U} : Y, MAC'_A$.

1) To start a new session, the authenticator U_A generates a message $\{ID_A, UID, X, C_0, MAC_A\}$ and sends it to the first user U_1 of the user group $\mathbb{U}$. In this message, ID_A is U_A's identity, UID is the identity set of all users in $\mathbb{U}$, parameter X carries some important information that U_A wants to deliver to all users in $\mathbb{U}$, C_0 is a parameter that is used to calculate C_1 and MAC_A is the message authentication code (MAC) [23] for message $\{ID_A, UID, X, C_0\}$.

   After receiving the message from U_A, U_1 first checks the integrity of the message. If the message is not tampered, U_1 continues. Otherwise, it aborts the session and the authentication fails.

2) If the message received by U_i ($1 \leq i \leq N-1$) is valid, U_i first computes C_i based on C_{i-1}, then generates a key parameter and adds it to KP_U, where KP_U is the key parameter set of user group $\mathbb{U}$. Next U_i computes the MAC value MAC_i of $\{ID_i, UID, X, KP_U, C_i\}$ and sends $\{ID_i, UID, X, KP_U, C_i, MAC_i\}$ to U_{i+1}. When U_{i+1} receives the message, it does the same as U_1 did in Step 1).

3) The behavior of U_N is the same as U_1. Once U_A verifies that the message from U_N has not been tampered with, it checks whether C_N is valid. If so, all users in user group $\mathbb{U}$ are successfully authenticated. If either MAC_N or C_N is invalid, U_A aborts the session and the group authentication fails.

4) U_A embeds key parameters generated by the user group in parameter Y. Meanwhile, U_A computes the session keys based on parameters from KP_U and those generated by itself. Then U_A computes MAC'_A and sends the whole message to every user in $\mathbb{U}$.

   After user U_i gets the message from U_A and validates MAC'_A, it retrieves its key parameter from Y and computes session keys.

2.3 DLP-based protocols

In this subsection, we explain how to compute parameters C_i ($0 \leq i \leq N$), X and Y of DLP-based protocols for both Type I and Type II proposed in [3]. The details are as follows.

1) C_0 is computed by U_A by $C_0 = \xi(r) = \xi(g^{r_A})$, where $r_A \in [1, p-1]$ is a random number, ξ is a message to be encrypted by Elgamal encryption algorithm [24]. Similarly, U_i ($2 \leq i \leq N$) computes C_i as $C_i = C_{i-1} \times r^{x_i} = \xi(r^{\sum_{t=1}^{i} x_t})$.

2) X is computed as a solution of $X \equiv V_i \bmod k_i$ ($1 \leq i \leq N$) by using Chinese reminder theorem (CRT) [25], where k_i is a secret shared between U_A and U_i. In the DLP-based protocol of Type I, $V_i = \{y_i \oplus K_G, y_i \oplus t_i, g^{m_i}, h_i\}$ and $h_i = H(ID_A \oplus ID_i \oplus y_A \oplus t_i)$ which is used to authenticate U_A by U_i. Here, y_i is a pre-shared secret between U_A and U_i, K_G is the group session key generated by U_A, t_i is a nonce and g^{m_i} is the key parameter generated by U_A to compute the key shared between U_A and U_i. In the DLP-based protocol of Type II, $V_i = SIGN_{SK_A}\{ID_A, ID_i, K_G, g^{m_i}, t_i\}$. Parameters g^{m_i} and t_i have the same meanings as in Type I, and the authentication of U_A is realized by the verification by its signature instead of using h_i as in Type I.

3) U_A computes Y by solving $Y \equiv W_i \bmod k_i$ ($1 \leq i \leq N$), where $W_i = \{ID_A, ID_i, KP_i\}$ and $KP_i = KP_U - \{g^{n_i}\}$.

4) The session key between U_A and U_i is computed as $g^{m_i n_i}$, while the session key between U_i and U_j ($1 \leq i, j \leq N$, $i \neq j$) is computed as $g^{n_i n_j}$.

3 Overview of Tool Scyther

In this section, we give a brief overview of model checking tool Scyther. We start with the presentation of the adversary model in Subsection 3.1 and the specification language used by Scyther specially in Subsection 3.2. We describe the claim specification and security requirements in Subsection 3.3.

3.1 Adversary model of Scyther

The adversary model used by Scyther is predefined and based on Dolev-Yao model [9]. It means that we do not need to formalize an adversary's abilities when we analyze protocols. An adversary (denoted by A) in Scyther can eavesdrop messages on the communication channel and can learn from the messages it has got. In the following, we explain how an adversary gains new knowledge.

Assume M is the adversary's knowledge set and f is a function to express the relations among different terms in M. k can represent both a symmetric and asymmetric key and k^{-1} is its reverse, while k^{-1} equals to k in case of a symmetric key. Let (t_i, t_j) represents the concatenation of terms t_i and t_j.

- $t \in M \Rightarrow M \vdash t$: if t is an element of M, A knows t.
- $M \vdash (t_1, t_2) \Rightarrow \{M \vdash t_1, M \vdash t_2\}$: if A knows (t_1, t_2), then A knows both terms t_1 and t_2.
- $\{M \vdash t_1, M \vdash t_2\} \Rightarrow M \vdash (t_1, t_2)$: if A knows both terms t_1 and t_2, A knows (t_1, t_2).
- $\bigwedge_{1 \leq i \leq n} M \vdash t_i \Rightarrow M \vdash f(t_1, \cdots, t_n)$: if A knows all t_i ($1 \leq i \leq n$) and f is a public function, then A can compute the result of function f with the input $t_1, \cdots, t_n$.
- $\{M \vdash t, M \vdash k\} \Rightarrow M \vdash \{t\}_k$: if A knows message t and key k, Ae can compute encrypted message $\{t\}_k$.

- $\{M \vdash \{t\}_k, M \vdash k^{-1}\} \Rightarrow M \vdash t$: if A knows the encrypted message $\{t\}_k$ and the decryption key k^{-1}, A can decrypt ciphertext and get plaintext t.

Therefore, an adversary A with the above learning abilities can eavesdrop messages on the communication channel and learn from these messages to expand its knowledge. In addition, A can delete messages on the communication channel, create new messages and insert them into the communication channel.

3.2 Specification language of Scyther

Scyther has its own specification language to describe protocols, roles, types of parameters, sending and receiving messages and so on. In the following (Example 1), we will explain the most important parameters and elements that we will use in our protocol formalization, including the definitions of predefined type, **usertype**, symmetric key, asymmetric keys, **hashfunction**, **role**, **protocol**, and message sending and receiving.

Example 1:

$usertype\ SharedSecret;$
$hashfunction\ H;$
$protocol\ Example(A,B)\{$
$\quad role\ A\{$
$\quad\quad fresh\ Na : SharedSecret;$
$\quad\quad var\ Nb, Nb' : Nonce;$
$\quad\quad send\_1(A,B,\{Na\}k(A,B));$
$\quad\quad recv\_2(B,A,Nb,Nb');$
$\quad \};$
$\quad role\ B\{$
$\quad\quad fresh\ Nb : Nonce;$
$\quad\quad var\ Na : SharedSecret;$
$\quad\quad recv\_1(A,B,\{Na\}k(A,B));$
$\quad\quad send\_2(B,A,Nb,\{H(Nb)\}sk(B));$
$\quad \};$
$\}$

Example 1 defines a **protocol** named *Example* where two communication parties A and B sending messages to each other. A communication party or an agent is declared as a **role**, where they are denoted by A and B in Example 1. *SharedSecret* is a user-defined type and it is declared by the term **usertype**, by which we can define different new types. The term **fresh** is a predefined type and is used to declare a value type that only exists in the session where it is generated. Term **nonce** is a predefined type used to describe a constant, for instance, Na defined within the domain of role A, and its value remain constant during the whole session. Term **var** is used to define a variable that is usually used to store a received value from the other party. For example, Na defined inside role B is a variable to receive the value sent from A rather than a constance such as Na defined in role A. The term **hashfunction** is used to define a hash function [26]. Its definition is usually global and all agents and protocols should have access to it. In Example 1, H is defined as the type of **hashfunction**, and the hash value of Nb is computed as

$H(Nb)$. There are two types of keys, symmetric and asymmetric. A symmetric key defined by $k(A,B)$ is a long-term value shared between A and B, and a message Na encrypted by it is described as $\{Na\}k(A,B)$. Asymmetric keys possessed by an agent B are a key pair, including a private key ($sk(B)$) and a public key ($pk(B)$). A message $H(Nb)$ signed by B can be denoted by $\{H(Nb)\}sk(B)$. Message sending and receiving in Scyther can be specified by the pair $send(s,r,m)$ and $recv(s,r,m)$, where s is a sender, r is a receiver and m is a message.

In the following, we use Example 2 to explain how to use a special role construction to express Diffie-Hellman key exchange protocol [27], which was used in the DLP-based protocols to establish session keys in the original protocols proposed in [3]. Compared with the role definition in Example 1, there are three differences in Example 2. First of all, the sender and the receiver are the same, which is denoted as "DH" here. Secondly, the messages sent and received are different. Finally, sending and receiving is expressed by $send\_!2()$ and $recv\_!1()$ rather than by $send\_2()$ and $recv\_1()$. By using this special structure, we intend to express that the computation results of $h(g(r),i))$ and $h(g(i),r))$ are equal. More similar examples can be found the Scyther manual [1].

Example 2:

$$
\begin{array}{l}
role\ DH\{ \\
\quad var\ i,r : Nonce; \\
\quad recv\_!1(DH,DH,h(g(r),i)); \\
\quad send\_!2(DH,DH,h(g(i),r)); \\
\}
\end{array}
$$

3.3 Events and claims

In this subsection, we describe how to formalize security requirements in Scyther, using **match** and **claim**. The event **match** can be used in two different ways. It can be used to specify equality constrains, for example, the codes after event $match(p_1, p_2)$ can only be executed if p_1 equals to p_2. The second usage of **match** is a value assignment, which is similar to "=" in C programming language. Assume p is a variable and v is a value, and $match(p,v)$ means assigning value v to variable p.

We use **claim** to specify security requirements **Alive**, **Nisynch**, **secret** and **commitment**. **Alive** is a form of authentication which aims to ensure that an intended communication party (R) has executed some events (e.g., $claim(R,Alive)$). **Nisynch** means that all received messages of R are indeed sent by the communication partner (sender) and have been received by another communication partner (receiver). It is expressed as $claim(R,Nisynch)$. If a term rt is claimed to be secret, rt should be kept secret to the adversary. More specifically, $claim(R,secret,rt)$ means that R claims that rt must be unknown to an adversary. If rt is a session key, we use $claim(R,SKR,rt)$ to specify it. **Commitment** is a promise of a communication partner to another party. For instance, $Claim(R,Commit,R',t)$ means that role R make a promise t to role R'. In this paper, we use **commitment** to verify protocols against impersonation attack.

4 Formal Analysis of the DLP-based Protocols of Type I

In this section, we describe analysis of the protocol of Type I when the number of group members N increases from 2 to 4. Later, we discuss the general situation when the number of group number is N ($N \geq 3$).

4.1 Formalization of security requirements

Assume there are two communication parties, i.e., R and R'. The DLP-based protocols are claimed to satisfy the following security requirements:

- **Mutual authentication** Authentication is way to ensure that a communication party is exchanging messages with an intended party. If authentication is achieved by both communication parties, it is called a mutual authentication. As described in Section 2, the authentication of the authenticator U_A by U_i in the DLP-based protocol of Type I can be confirmed if h_i' equals to h_i. However, the whole group can be considered authenticated only if C_N' equals to C_N. We will use **match** to check the equality of C_N and C_N'. In addition, the security property **Alive** will also be required, to make sure that it is the intended communication parties rather than someone else.

- **Implicit key authentication** If a protocol satisfies implicitly key (k) authentication [3] and R claims this security requirement, it means that R' is the only entity who has the possibility to possess this k. In this paper, we use $claim(R, SKR, k)$ and $claim(R', SKR, k)$ to express it.

- **Secure against impersonation attack** Impersonation attack is an attack where an adversary behaves under the identity of a legitimate communication party. Therefore, this security requirement can be inferred from mutual authentication. As long as mutual authentication holds, we can claim that none of the communication parties is impersonated by an adversary. As discussed before, this can be ensured by checking whether $h_i' = h_i$ and $C_N' = C_N$ hold.

- **Secure against passive adversaries** A passive adversary eavesdrops messages on the communication channel, analyzes these messages and tries to learn as much as possible. Compared with an active adversary, the abilities of a passive one are limited. It cannot delete or insert messages into the communication channel, and its main goal is to learn **useful** information from the messages that it has eavesdropped. In the DLP-based protocol of Type I, the most useful information is the group key (k) and session keys (k), and we can use $claim(R, SKR, k)$ to express it.

- **Provide forward secrecy and backward secrecy** If a protocol provides forward secrecy, the exposure of keys in current session will not lead to the exposure of keys of future sessions. If a protocol provides backward secrecy, the compromise of keys in current session will not cause the compromise of session keys of past sessions. Since Scyther does not support long term values except for the shared key between two parties, we will not analyze these two security requirements in this paper.

4.2 Specification of difficult problems

In this subsection, we specify difficult mathematical problems in DLP-based protocols of Type I, including Diffie-Hellman key exchange [27], hash functions, Chinese remainder theorem [25], proxy encryption [24], session key computation, MAC and pre-shared values.

- **Diffie-Hellman key exchange (session key computation), hash functions, proxy encryption, MAC** As described in Subsection 3.2, the type **hashfunction** is used to declare a secure hash function, which is a one-way function (Its inverse is infeasible to compute). Therefore, difficult mathematical problems, such as Diffie-Hellman problem used to compute session keys,cryptographic hash functions, proxy encryption and MAC can be considered as one-way hash functions, because an adversary defined by Scyther cannot compute their inverse.

  If two parties A and B want to establish a session key based on Diffie-Hellman key exchange, they should generate parameters a and b first, and send g^a and g^b to each other. Then A and B

compute their session keys as $(g^b)^a$ and $(g^a)^b$ respectively. To formalize it, we first declare two **hashfucntion** g and h and then express the session keys as $h(g(b),a)$ and $h(g(a),b)$. According to Example 2 in Subsection 3.2, we have $h(g(b),a) = h(g(a),b)$. Similarly, we use **hashfuntion** H, C and MAC to specify hash functions, proxy encryption and MAC.

• **Chinese remainder theorem (CRT)** In the original protocols [3], parameters X and Y are computed by $X \equiv V_i \bmod k_i$ $(1 \leq i \leq N)$ and $Y \equiv W_i \bmod k_i$ $(1 \leq i \leq N)$ using CRT, where k_i is a long term value shared between U_A and U_i. However, Scyther does not support long-term value except for the symmetric and asymmetric keys. Since only symmetric keys rather than asymmetric keys are shared between two parties, we will use a symmetric key between U_A and U_i to simulate this long-term value k_i.

• **Pre-shared secrets** As described in Subsection 2.3, x_i $(1 \leq i \leq N)$ is another long-term shared value, and it is used for mutual authentication. Since the symmetric key $k(U_A, U_i)$ between U_A and U_i has already been used to simulate k_i, we need to formalize x_i differently. The main idea is as follows. Since x_i is a long term value used for mutual authentication, it should be enough to assume that x_i has already been shared between U_A and U_i before the mutual authentication. Therefore, we will embed x_i in X. Since the parameters in V_i can only be extracted by U_i, this assumption is reasonable and realistic.

4.3 Specification of the protocols with different group sizes

In our experiments, we have formalized DLP-based protocols when the number of group member is two, three and four to check whether the security requirements of mutual authentication, implicit key authentication, secure against impersonation attack and passive adversaries. In Listing 3, we show part of the specification codes to explain how to formalize the protocol when $N = 2$.

Listing 1: Type I protocol specification for 2 group members

```
#Type definitions
hashfunction g, h;
hashfunction C, H, MAC;
usertype mtype, gtype, htype, ctype, wtype;

protocol Group-authentication-DLP(UA , U1, U2)
{
  ...
  macro v1 = {KG, x1,  t1, gm1, h1}k(UA, U1);
  macro v2 = {KG, x2,  t2, gm2, h2}k(UA, U2);
  macro w1 = {UA, U1, gn2}KG;
  macro w2 = {UA, U2, gn1}KG;
  role UA
  {
    ...
    match(gm1, g(m1));    ...
    match(h1, H(UA, U1,  xa,  t1));  ...
    match(MACA11, MAC(KG, UA, U1, U1, U2, v1, r, C0));  ...
    send_1(UA, U1, U1, U2, v1, r, C0, MACA11);
    send_2(UA, U2, U1, U2, v2, r, MACA12);
    recv_4(U2, UA,  gn1, gn2, C2, MAC2);    ...
    match (MAC2, MAC2');  ...
    match (C2, C2');
    send_5(UA, U1, w1, MACA21);
    send_6(UA, U2, w2, MACA22);
    #check security requirements
    claim(UA, Alive);
    claim(UA, Nisynch);
    claim(UA, SKR, KG);
    claim(UA, SKR, h(gn1, m1));
```

```
31     claim(UA, SKR, h(gn2, m2));
32   }
33   role U1
34   {
35     ....
36     recv_1(UA, U1, U1, U2, v1, r, C0, MACA11);    ...
37     match(MACA11, MACA11')
38     match(h1, h1'); ...
39     send_3(U1, U2,  gn1, C1, MAC1);
40     recv_5(UA, U1, w1, MACA21);    ...
41     match(MACA21, MACA21'); ...
42     claim(U1, Alive);
43     claim(U1, Nisynch);
44     claim(U1, SKR, KG);
45     claim(U1, SKR, h(gm1, n1));
46     claim(U1, SKR, h(gn2, n1));
47   }
48   role U2
49   {
50     ...
51     recv_2(UA, U2, U1, U2, v2, r, MACA12);   ...
52     match(MACA12, MACA12');
53     match(h2, h2'); ...
54     send_4(U2, UA,  gn1, gn2, C2, MAC2);     ...
55     recv_6(UA, U2, w2, MACA22);    ...
56     match(MACA22, MACA22'); ...
57     claim(U2, Alive);
58     claim(U2, Nisynch);
59     claim(U2, SKR, KG);
60     claim(U2, SKR, h(gm2, n2));
61     claim(U2, SKR, h(gn1, n2));
62   }
63   role DH{
64     var i, r: Nonce;
65     recv_!1(DH, DH, h(g(r), i));
66     send_!2(DH, DH, h(g(i), r));
67   }
68 }
```

As shown above, the authenticator has to formalize $v1$, v_2, w_1 and w_2 first. They are declared as global such that all roles have access to them. Before sending out messages to users U_1 and U_2, U_A has to prepare parameters g^{m_1}, g^{m_2}, h_1, h_2 for v_1 and v_2, and compute all necessary MACs. All these preparations are finished before line 18. In lines 19 and 20, U_A sends out the first two messages to U_1 and U_2. U_1 and U_2 receive these two messages and check the validity of MAC and the quality of h_i, $i \in \{1,2\}$, by using event *match*. If these checks are successful, the authentication of U_A by both U_1 and U_2 succeeds. After checking *MAC* of message one and h_1 from U_A, U_1 calculates C_1 and sends the third message to U_2 in line 39. U_2 receives message three, computes C_3 and then sends message four to U_A. Once U_A receives the message from U_2 and the message is not tampered, it checks the equality of C_N and C'_N. If $C_N = C'_N$ holds, both U_1 and U_2 are authenticated. In message five and six, U_A sends out session key parameters to U_1 and U_2. At last, we check security requirements using five *claim*s as described in Subsection 4.1. The experiment result is shown in Fig. 2.

We have also carried out experiments when $N = 3$ and $N = 4$ and checked the same security requirements. Results show that all checked security requirements (mutual authentication, implicit key authentication, security against impersonation attack and passive adversaries) are satisfied. Based on these experience, we show how to specify the DLP-based protocols of Type I for arbitrary finite N ($N \geq 3$) group members in Listing 2.

Claim				Status		Comments
Group_authentication_DLP	UA	Group_authentication_DLP,UA1	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,UA2	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,UA3	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,UA4	SKR h(gn1,m1)	Ok	Verified	No attacks.
		Group_authentication_DLP,UA5	SKR h(gn2,m2)	Ok	Verified	No attacks.
	U1	Group_authentication_DLP,U11	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,U12	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,U13	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,U14	SKR h(gm1,n1)	Ok	Verified	No attacks.
		Group_authentication_DLP,U15	SKR h(gn2,n1)	Ok	Verified	No attacks.
	U2	Group_authentication_DLP,U21	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,U22	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,U23	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,U24	SKR h(gm2,n2)	Ok	Verified	No attacks.
		Group_authentication_DLP,U25	SKR h(gn1,n2)	Ok	Verified	No attacks.

Figure 2: Experiment results of the DLP-based protocols of Type I for 2 group members

Listing 2: Type I protocol specification for N group members

```
protocol Group-authentication-DLP1(UA, U1, ..., UN)
{
  ...
  role UA
  {
    ...
    send_i(UA, Ui, U1, ..., UN, vi, r, C0, MACA1i);
    ...
    recv_2N(UN, UA, gn1, ..., gnN, CN, MACAN); ...
    match(MACN, MACN');
    match(CN, CN');    ...
    send_(2N+i)(UA, Ui, wi, MACA2i);
    ...
    claim(UA, Alive); ...
    claim(UA, h(gn1, m1));
    ...
    claim(UA, h(gnN, mN));
  }
  role U1
  {
    ...
    recv_1(UA, U1, U1, ..., UN, v1, r, C0, MACA11);    ...
    match(h1, h1');
```

```
    match(MACA11,MAC11'); ...
    send_(N+1)(U1, U2, gn1, C1, MAC1);
    recv_(2N+1)(UA, U1, w1, MACA21);    ...
    match(MAC21, MAC21');
    claim(U1, Alive); ...
    claim(U1, h(gm1, n1));
    claim(U1, h(gn2, n1));
    ...
  }
  role Ui #for users from U2 to U(N-1)
  {
    ...
    recv_i(UA, Ui, U1, ..., UN, vi, r, MACA1i); ...
    match(MACA1i, MACA1i');
    match(hi, hi'); ...
    recv_(N+i-1)(U(i-1), Ui, gn1, ..., gn(i-1), C(i-1), MAC(i-1)); ...
    match(MAC(i-1), MAC(i-1)'); ...
    send_(N+i)(Ui, U(i+1), gn1, ..., gni, Ci, MACi);    ...
    recv_(2N+i)(UA, Ui, wi, MACA2i);    ...
    match(MACA2i, MACA2i');
    claim(Ui, Alive); ...
    claim(Ui, h(gmi, ni));
    ...
    claim(Ui, h(gn(i-1), ni));
    claim(Ui, h(gn(i+1), ni));
    ...
  }
  role UN
  {
    ...
    recv_N(UA, UN, U1, ..., U3, vN, r, MACA1N); ...
    match(MACA1N, MACA1N');
    match(hN, hN'); ...
    recv_(2N-1)(U(N-1), UN, gn1, ..., gn(N-1), C(N-1), MAC(N-1));    ...
    match(MAC(N-1), MAC(N-1)');
    send_2N(UN, UA, gn1, ..., gnN, CN, MACN);    ...
    recv_3N(UA, UN, wN, MACA2N);    ...
    match(MACA2N, MACA2N');
    claim(UN, Alive);   ...
    claim(UN, h(gmN, nN));
    ...
    claim(UN, h(gm(N-1), nN));
  }
  role DH{
    var i, r: Nonce;
    recv_!1(DH, DH, h(g(r), i));
    send_!2(DH, DH, h(g(i), r));
  }
}
```

In Listing 2, we have only described the message flow, MACs verification, equality checking and security requirement verification. However, details on how to prepare parameters for the messages are omitted since they have already be discussed in the case of two group users. The message flow contains three parts: N messages from U_A to all U_i in the user group to deliver V_i; messages from U_1 to U_2, U_2 to U_3 and so on until the message from U_N to U_A. Finally, N messages from U_A to U_i to deliver key parameters to compute session keys. The integrity of every received message must be checked by verifying its MAC and it is realized by the event *match*. In addition, the equality checking of h_i and C_N is also expressed by event *match*. Several *claims* are used to express and check the security requirements, such as "Alive" and the secrecy of session keys. The secrecy of session keys analyzes of the group key K_G, session keys between U_A and U_i and the session keys between different group members.

5 Formal Analysis of the DLP-based Protocols of Type II

The specification of the DLP-based protocols of Type II is similar to Type I. More specifically, the formalization of difficult problems and all security requirements except for mutual authentication are the same. Compared with the formalization of the DLP-based protocols of Type I, there are mainly two differences. First, when U_A sends out messages which include V_i to all group members in protocols of Type I, h_i is included for later mutual authentication. However, in protocols of Type II, U_A uses its signature instead of h_i for its authentication. Accordingly, when group users receives these messages from U_A, they do not need to check the equality of h_i to finish the authentication of U_A. Instead, they verify U_A's signature. This property can be provided by the security of PKI based signatures and thus we do not have to check it here. Based on the above differences, we give the specification how to formalize the DLP-based protocols of Type II when the number of group users is N ($N \geq 3$) (Listing 3).

The presented specification (Listing 3) shows how to specify the DLP-based protocols of Type II when the group members are two, three and four and how to verify the security requirements including mutual authentication, implicit key authentication, security against impersonation attack and passive adversaries. Results (Fig. 3 is the result for two group members.) show that all these four security requirements are satisfied when the number of group members varies from two to four.

Claim				Status		Comments
Group_authentication_DLP	UA	Group_authentication_DLP,UA1	Commit U1,{{UA,U1,KG,x1,t1,gm1}sk(UA)}k(UA,U1)	Ok	Verified	No attacks.
		Group_authentication_DLP,UA2	Commit U2,{{UA,U2,KG,x2,t2,gm2}sk(UA)}k(UA,U2)	Ok	Verified	No attacks.
		Group_authentication_DLP,UA3	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,UA4	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,UA5	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,UA6	SKR h(gn1,m1)	Ok	Verified	No attacks.
		Group_authentication_DLP,UA7	SKR h(gn2,m2)	Ok	Verified	No attacks.
	U1	Group_authentication_DLP,U11	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,U12	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,U13	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,U14	SKR h(gm1,n1)	Ok	Verified	No attacks.
		Group_authentication_DLP,U15	SKR h(gn2,n1)	Ok	Verified	No attacks.
	U2	Group_authentication_DLP,U21	Alive	Ok	Verified	No attacks.
		Group_authentication_DLP,U22	Nisynch	Ok	Verified	No attacks.
		Group_authentication_DLP,U23	SKR KG	Ok	Verified	No attacks.
		Group_authentication_DLP,U24	SKR h(gm2,n2)	Ok	Verified	No attacks.
		Group_authentication_DLP,U25	SKR h(gn1,n2)	Ok	Verified	No attacks.

Figure 3: Experiment results of the DLP-based protocols of Type II for 2 group members

Listing 3: Type II protocol specification for N group members

```
protocol Group−authentication−DLP2(UA, U1, ..., UN)
{
  ...
  macro vi = {{UA, Ui, KG, xi, ti, gmi}sk(UA)}k(UA, Ui);
  macro wi = {UA, Ui, gn1, ..., gn(i−1), gn(i+1), ..., gnN}KG;
  ...
  role UA
  {
    ...
    claim(UA, Commit, Ui, vi); #different from Type 1
    send_i(UA, Ui, U1, ..., UN, vi, r, C0, MACA1i);
    ...
    recv_2N(UN, UA, gn1, ..., gnN, CN, MACN);    ...
    match(MACN, MACN');    ...
    send_(2N+1)(UA, U1, w1, MACA21);
    ...
    send_3N(UA, UN, wN, MACA2N);
    claim(UA, Alive); ...
    claim(UA, h(gn1, m1));
    ...
    claim(UA, h(gnN, mN));
  }
  role U1
  {
    ...
    recv_1(UA, U1, U1, ..., UN, v1, r, C0, MACA11);
    match(MACA11, MACA11'); #no need to check h1 here
    ...
    send_(N+1)(U1, U2, gn1, C1, MAC1);
    recv_(2N+1)(UA, U1, w1, MACA21);    ...
    match(MACA21, MACA21');    ...
    claim(U1, Alive); ...
    claim(U1, h(gm1, n1));
    claim(U1, h(gn2, n1));
    ...
  }
  role Ui #for users from U2 to U(N−1)
  {
    ...
    recv_i(UA, Ui, U1, ..., UN, vi, r, MACA1i); ...
    match(MACA1i, MACA1i');    ...    #no need to check hi
    recv_(N+i−1)(U(i−1), Ui, gn1, ..., gn(i−1), C(i−1), MAC(i−1)); ...
    match(MAC(i−1), MAC(i−1)'); ...
    send_(N+i)(Ui, U(i+1), gn1, ..., gni, Ci, MACi);    ...
    recv_(2N+i)(UA, Ui, wi, MACA2i);    ...
    match(MACA2i, MACA2i');    ...
    claim(Ui, Alive); ...
    claim(Ui, h(gmi, ni));
    ...
    claim(Ui, h(gn(i−1), ni));
    claim(Ui, h(gn(i+1), ni));
    ...
  }
  role UN
  {
    ...
    recv_N(UA, UN, U1, ..., U3, vN, r, MACA1N); ...
    match(MACA1N, MACA1N'); ... #no need to check hN
    recv_(2N−1)(U(N−1), UN, gn1, ..., gn(N−1), C(N−1), MAC(N−1));    ...
    match(MAC(N−1), MAC(N−1)');    ...
    send_2N(UN, UA, gn1, ..., gnN, CN, MACN);    ...
    recv_3N(UA, UN, wN, MACA2N);    ...
    match(MACA2N, MACA2N');    ...
    claim(UN, Alive);    ...
    claim(UN, h(gmN, nN));
    ...
```

```
    claim(UN, h(gm(N-1), nN));
  }
  ...
}
```

6 Conclusions

This paper is an extension of the conference paper [20]. In [20], we used the model checking Scyther to analyze DLP-based group authentication protocols proposed in [3] and checked four security requirements, i.e., mutual authentication, implicit key authentication, security against impersonation attack and passive adversaries, for the case of three members in the user group. In this paper, we present analysis of the DLP-based protocols for two and four group members, in addition to three group members. Results show that the protocols satisfy the same four security requirements as for three members. Compared with the work in [20], the most important innovation in this paper is that we have provided general design (Listing 2 and Listing 3), based on which we can construct models that specify DLP-based protocols of both types for the group number of size N ($N \geq 3$).

References

[1] C. Cremers, "The scyther tool," www.cs.ox.ac.uk/people/cas.cremers/scyther/ [Online; Accessed on June 10, 2016].

[2] R. Zunino and P. Degano, "A note on the perfect encryption assumption in a process calculus," in *Proc. of the 7th International Conference of Foundations of Software Science and Computation Structures (FOSSACS'04), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'04), Barcelona, Spain*, ser. Lecture Notes in Computer Science, vol. 2987. Springer Berlin Heidelberg, March-April 2004, pp. 514–528.

[3] H. Yang, L. Jiao, and V. A. Oleshchuk, "A general framework for group authentication and key exchange protocols," in *The 6th International Symposium of Foundations and Practice of Security (FPS'13), La Rochelle, France, October 21-22, 2013, Revised Selected Papers*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8352, pp. 31–45.

[4] N. Koblitz and J. A. Menezes, "Another look at "provable security"," *Journal of Cryptology*, vol. 20, no. 1, pp. 3–37, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00145-005-0432-z

[5] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270 – –299, 1984.

[6] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[7] P. Wolper, "The meaning of "formal": from weak to strong formal methods," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1, pp. 6–8, 2014. [Online]. Available: http://dx.doi.org/10.1007/s100090050002

[8] C. Cremers, S. Mauw, and E. de Vink, "Formal methods for security protocols: Three examples of the black-box approach," *NVTI newsletter*, vol. 7, pp. 21–32, 2003.

[9] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, March 1983.

[10] SPIN, "Verifying multi-threaded software with spin," http://spinroot.com/spin/whatispin.html [Online; Accessed on June 10, 2016].

[11] G. Holzmann, *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley, 2003.

[12] P. Lafourcade, V. Terrade, and S. Vigier, "Comparison of cryptographic verification tools dealing with algebraic properties," in *The 6th International Workshop of Formal Aspects in Security and Trust (FAST'09), Eindhoven, The Netherlands, November 5-6, 2009, Revised Selected Papers*, ser. Lecture Notes in Computer Science, P. Degano and J. D. Guttman, Eds. Springer Berlin Heidelberg, 2010, vol. 5983, pp. 173–185.

[13] C. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Proc. of the 20th International Conference of Computer Aided Verification (CAV'08), Princeton, NJ, USA*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, July 2008, pp. 414–418.

[14] C. Cremers, "Key exchange in ipsec revisited: Formal analysis of IKEv1 and IKEv2," in *Proc. of the 16th European Conference on Research in Computer Security (ESORICS'11), Leuven, Belgium*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 315–334.

[15] D. Basin, C. Cremers, and S. Meier, "Provably repairing the ISO/IEC 9798 standard for entity authentication," in *Proc. of the 1st International Conference of Principles of Security and Trust (POST'12), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS'12), Tallinn, Estonia*, ser. Lecture Notes in Computer Science, vol. 7215. Springer Berlin Heidelberg, March-April 2012, pp. 129–148.

[16] D. Basin and C. Cremers, "Evaluation of ISO/IEC 9798 protocols version 2.0," April 2011, https://www.cs.ox.ac.uk/people/cas.cremers/downloads/papers/BC2011-iso9798-v2.pdf [Online; Accessed on June 10, 2016].

[17] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *Proc. of the 15th European Symposium on Research in Computer Security (ESORICS'10), Athens, Greece*, ser. Lecture Notes in Computer Science, vol. 6345. Springer Berlin Heidelberg, September 2010, pp. 340–356.

[18] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol," in *Advances in Cryptology – Proc. of the the 25th Annual International Cryptology Conference (CRYPTO'05), Santa Barbara, California, USA, August 14-18, 2005*, ser. Lecture Notes in Computer Science, vol. 3621. Springer Berlin Heidelberg, August 2005, pp. 546–566.

[19] B. Ustaoglu, "Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS," *Designs, Codes and Cryptography*, vol. 46, no. 3, pp. 329–342, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10623-007-9159-1

[20] H. Yang, A. Prinz, and V. Oleshchuk, "Formal analysis and model checking of a group authentication protocol by Scyther," in *Proc. of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'16), Heraklion, Greece*. IEEE, February 2016, pp. 553–557.

[21] J. Zhang and L. Chen, “An improved algorithm for discrete logarithm problem,” in *Proc. of the 2009 International Conference on Environmental Science and Information Application Technology (ESIAT’09), Wuhan, China*, vol. 2. IEEE, July 2009, pp. 658–661.

[22] D. Hankerson and A. Menezes, “Elliptic curve discrete logarithm problem,” in *Encyclopedia of Cryptography and Security*. Springer US, 2005, pp. 186–189. [Online]. Available: http://dx.doi.org/10.1007/0-387-23483-7_132

[23] D. H. Van and N. D. Thuc, “A privacy preserving message authentication code,” in *Proc. of the 5th International Conference on IT Convergence and Security (ICITCS’15), Kuala Lumpur, Malaysia*. IEEE, August 2015, pp. 1–4.

[24] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, Jul 1985.

[25] Y. Wang, “New chinese remainder theorems,” in *Conference Record of the 32nd Asilomar Conference on Signals, Systems & Computers (ACSSC’98), Pacific Grove, CA, USA*, vol. 1. IEEE, November 1998, pp. 165–171.

[26] B. Preneel, “Cryptographic hash functions: Theory and practice,” in *Progress in Cryptology - Proc. of the 11th International Conference on Cryptology in India (INDOCRYPT’10), Hyderabad, India*, ser. Lecture Notes in Computer Science, vol. 6498. Springer Berlin Heidelberg, December 2010, pp. 115–117. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-17401-8_9

[27] D. Boneh, “The decision diffie-hellman problem,” in *Proc. of the 3rd International Symposium of Algorithmic Number Theory (ANTS-III), Portland, Oregon, USA*, ser. Lecture Notes in Computer Science, vol. 1423. Springer Berlin Heidelberg, June 1998, pp. 48–63.

Author biography

Huihui Yang received her B.S degree in computer science from Wuhan University, Wuhan, China, in 2007 and M.S in information security from the Chinese Academy of Science, Beijing, China in 2011 respectively. From February 2011 to April 2012, she worked as a software engineer in Ericsson, Beijing, China. Currently, she is currently a PhD candidate in the department of information and communication in University of Agder. Her research interests are security and privacy, applied cryptography, secure protocols and formal methods.

Vladimir Oleshchuk is Professor of Computer Science and Head of the Communication and System Security Group at the University of Agder, Norway. He received his MSc in Applied Mathematics (1981) and PhD in Computer Science (1988) from the Taras Shevchenko Kiev National University, Kiev, Ukraine, and his MSc in Innovations and Entrepreneurship from the Norwegian University of Science and Technology (NTNU). He is a senior member of the IEEE and a senior member of the ACM. His current research interests include security, privacy and trust for wireless systems and their applications to e-health, wireless sensor networks, P2P systems, and mobile systems, application of formal methods to enforce security and privacy, security-related text analysis and privacy-preserving data analysis.

Andreas Prinz holds an M.Sc. in mathematics and a Ph.D. in computer science from Humboldt-University Berlin. He had several positions at academic and industrial institutions in Germany and Australia. Since 2003, he is professor at University of Agder in Norway, where he was Head of the ICT department from 2007-2015. His research interests and competence include systems engineering with particular focus on modelling, languages and formal methods. Prof. Prinz has worked in several projects dealing with the development of modern ICT systems using advanced technology.