Yahalom-BAN simplified の ProVerif による評価結果

国立研究開発法人 情報通信研究機構

1. 基本情報

✧ 名前

BAN simplified version of Yahalom

✧ 機能

共通鍵暗号の鍵サーバを用いた相互認証・鍵交換プロトコル。

✧ 関連する文書

M.Burrows, M.Abadi, and R.Needham, “A Logic of Authentication,” Research Report 39, Digital Equipment Corp. Systems Research Center, Feb. 1989.

2. ProVerif の文法による記述

2.1. プロトコル仕様

```
(*
A, B, S :          principal
Na, Nb :           number fresh
Kas, Kbs, Kab :    key

A knows :          A, B, S, Kas
B knows :          B, S, Kbs
S knows :          S, A, B, Kas, Kbs

1.        A        ->        B        :        A, Na
2.        B        ->        S        :        B, Nb, {A, Na}Kbs
3.        S        ->        A        :        Nb, {B, Kab, Na}Kas, {A,
Kab, Nb}Kbs
4.        A        ->        B        :        {A,      Kab,      Nb}Kbs,
{Nb}Kab
*)
```

```
free c: channel.
type host.
type nonce.
type key.

table keys(host, key).

fun encrypt(bitstring, key): bitstring.
reduc forall x: bitstring, k: key; decrypt(encrypt(x, k), k) = x.

fun nonce_to_bitstring(nonce): bitstring [data,typeConverter].

free hostA, hostB: host.



free secretTest, secretA_Kab, secretB_Kab: bitstring [private].

event beginAnonce(host, host, nonce).
event beginBnonce(host, host, nonce, nonce).
event beginA(host, host, nonce, nonce, key).
event beginB(host, host, nonce, nonce, key).
event endA(host, host, nonce, nonce, key).
event endB(host, host, nonce, nonce, key).
event end().

query attacker(secretA_Kab).
query attacker(secretB_Kab).
(*
query attacker(secretTest).
*)

(*
```

```
query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endA(A, B, Na, Nb, Kab)) ==>
      inj-event(beginB(A, B, Na, Nb, Kab)).
*)

query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endB(A, B, Na, Nb, Kab)) ==>
      inj-event(beginA(A, B, Na, Nb, Kab)).

query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endA(A, B, Na, Nb, Kab)) ==>
      inj-event(beginBnonce(A, B, Na, Nb)).

query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endB(A, B, Na, Nb, Kab)) ==>
      inj-event(beginAnonce(A, B, Na)).

let processA(hostA: host, hostB: host) =
    in(c, (A: host, B: host));
    if A = hostA || A = hostB then
    (* 1 *)
    new Na: nonce;
    event beginAnonce(A, B, Na);
    out(c, (A, Na));
    (* 3 *)
    in(c, (Nb: nonce, c1: bitstring, c2: bitstring));
    get keys(=A, Kas: key) in
    let (=B, Kab: key, =Na) = decrypt(c1, Kas) in
    event beginA(A, B, Na, Nb, Kab);
    out(c, (c2, encrypt(nonce_to_bitstring(Nb), Kab)));
    (* security description *)
    if B = hostA || B = hostB then
```

```
    event endA(A, B, Na, Nb, Kab);
    out(c, encrypt(secretA_Kab, Kab));
    event end().

let processB(hostA: host, hostB: host) =
    in(c, (B: host));
    if B = hostA || B = hostB then
    (* 1 *)
    in(c, (A: host, Na: nonce));
    (* 2 *)
    new Nb: nonce;
    get keys(=B, Kbs: key) in
    event beginBnonce(A, B, Na, Nb);
    out(c, (B, Nb, encrypt((A, Na), Kbs)));
    (* 4 *)
    in(c, (c2: bitstring, c3: bitstring));
    let (=A, Kab: key, =Nb) = decrypt(c2, Kbs) in
    event beginB(A, B, Na, Nb, Kab);
    if nonce_to_bitstring(Nb) = decrypt(c3, Kab) then
    if A = hostB || A = hostA then
    event endB(A, B, Na, Nb, Kab);
    out(c, encrypt(secretB_Kab, Kab));
    event end().

let processS =
    (* 2 *)
    in(c, (B: host, Nb: nonce, c0: bitstring));
    get keys(=B, Kbs: key) in
    let (A: host, Na: nonce) = decrypt(c0, Kbs) in
    (* 3 *)
    get keys(=A, Kas: key) in
    new Kab: key;
```

```
    out(c, (Nb, encrypt((B, Kab, Na), Kas), encrypt((A, Kab, Nb), Kbs))).

let key_registration =
    in(c, (X: host, Kxs: key));
    if X <> hostA && X <> hostB then
    insert keys(X, Kxs).

process new Kas: key;
         insert keys(hostA, Kas);
         new Kbs: key;
         insert keys(hostB, Kbs);
         ((!processA(hostA, hostB)) | (!processB(hostA, hostB)) |
(!processS) |
          (!key_registration))
```

2.2. 攻撃者モデル

上述の記述に含まれる。

2.3. セキュリティ要件

上述の記述の (* queries *) に該当する。

```
(* (1) *)
query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endB(A, B, Na, Nb, Kab)) ==>
      inj-event(beginA(A, B, Na, Nb, Kab)).

(* (2) *)
query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endA(A, B, Na, Nb, Kab)) ==>
      inj-event(beginBnonce(A, B, Na, Nb)).

(* (3) *)
query A: host, B: host, Na: nonce, Nb: nonce, Kab: key;
      inj-event(endB(A, B, Na, Nb, Kab)) ==>
      inj-event(beginAnonce(A, B, Na)).
```

```
(* (4) *)
query attacker(secretA_Kab).
query attacker(secretB_Kab).
```

(1) 参加者Bによる参加者Aの認証と、ノンスNa, ノンスNb及びセッション鍵Kabの一致。

(2) 参加者Aによる参加者Bの認証と、ノンスNa及びノンスNbの一致。

(3) 参加者Bによる参加者Aの認証と、ノンスNa及びノンスNbの一致。

(4) セッション鍵Kabの秘匿性。

なお、「参加者Aによる参加者Bの認証と、ノンスNa, ノンスNb及びセッション鍵Kabの一致」は成り立たないことが自明なので評価を行なわなかった。

3. ProVerifによる評価結果

3.1. 出力

セキュリティ要件のうち、(4)のみ満たされる（true)、それ以外には攻撃が発見されたことを確認した。

```
RESULT inj-event(endB(A_32,B_33,Na_34,Nb_35,Kab_36)) ==>
inj-event(beginAnonce(A_32,B_33,Na_34)) is false.
RESULT (even event(endB(A_17030,B_17031,Na_17032,Nb_17033,Kab_17034))
==> event(beginAnonce(A_17030,B_17031,Na_17032)) is false.)
RESULT inj-event(endA(A_17305,B_17306,Na_17307,Nb_17308,Kab_17309))
==> inj-event(beginBnonce(A_17305,B_17306,Na_17307,Nb_17308)) is
false.
RESULT (even event(endA(A_32930,B_32931,Na_32932,Nb_32933,Kab_32934))
==> event(beginBnonce(A_32930,B_32931,Na_32932,Nb_32933)) is false.)
RESULT inj-event(endB(A_33107,B_33108,Na_33109,Nb_33110,Kab_33111))
==> inj-event(beginA(A_33107,B_33108,Na_33109,Nb_33110,Kab_33111)) is
false.
RESULT (even event(endB(A_49656,B_49657,Na_49658,Nb_49659,Kab_49660))
==> event(beginA(A_49656,B_49657,Na_49658,Nb_49659,Kab_49660)) is
false.)
RESULT not attacker(secretB_Kab[]) is true.
RESULT not attacker(secretA_Kab[]) is true.
```

3.2. 攻撃の解説

セキュリティ要件(3)に関する攻撃は、(1)に関する攻撃にもなっているため、(2)及び(3)への攻撃について述べる。(2)への攻撃を下図に示す。この攻撃では、参加者 A が暗号プロトコルにおける参加者 A の役割及び参加者 B の役割を同時に実行する。図中ではこの 2 つの役割をそれぞれ A_Init 及び A_Resp で表す。A_Init は攻撃者 Atk からメッセージを受けとった時点で相手は参加者 B であるとして認証を完了するが、実際には参加者 B は通信を行なっていない。

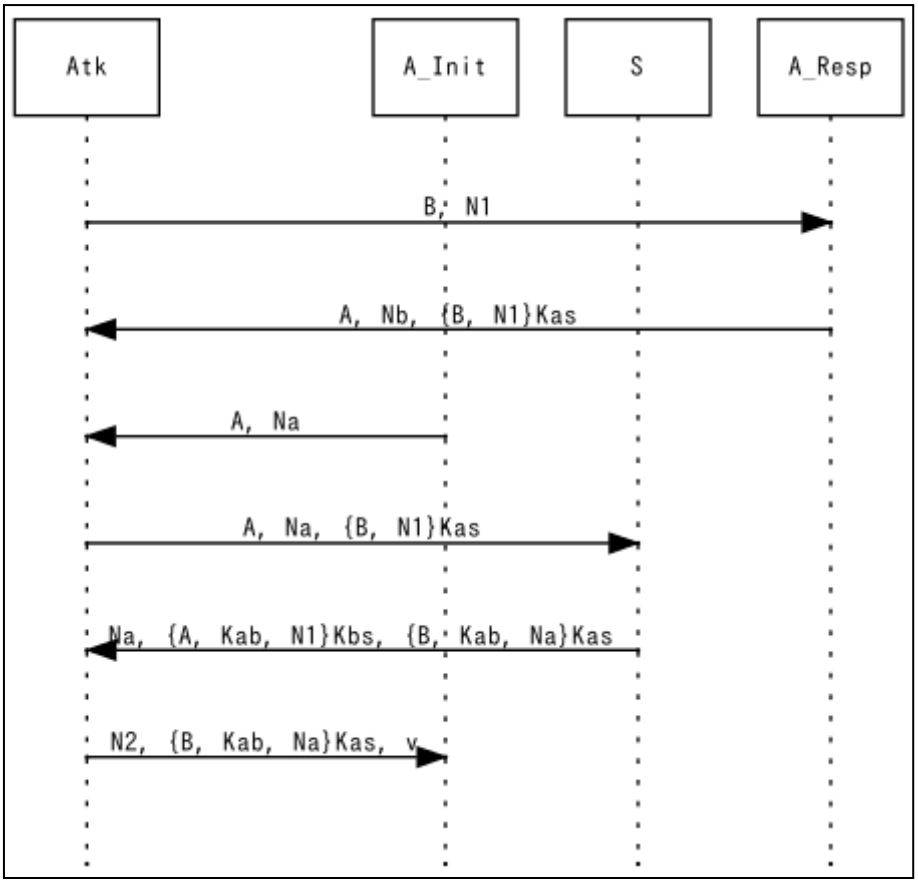

図. 1.　攻撃シーケンス

次に、(3)への攻撃を次ページの図 2. に示す。この攻撃では、参加者 A が暗号プロトコルにおける参加者 A の役割を実行し、参加者 B が暗号プロトコルにおける参加者 A 及び参加者 B の役割を同時に実行する。図中ではこれらの役割をそれぞれ A_Init、B_Init 及び B_Resp で表す。B_Resp は攻撃者 Atk から最後のメッセージを受けとった時点で相手は参加者 A であり、参加者 A が生成したノンスは N であるとして認証を完了するが、実際には参加者 A はノンス N を生成していない。

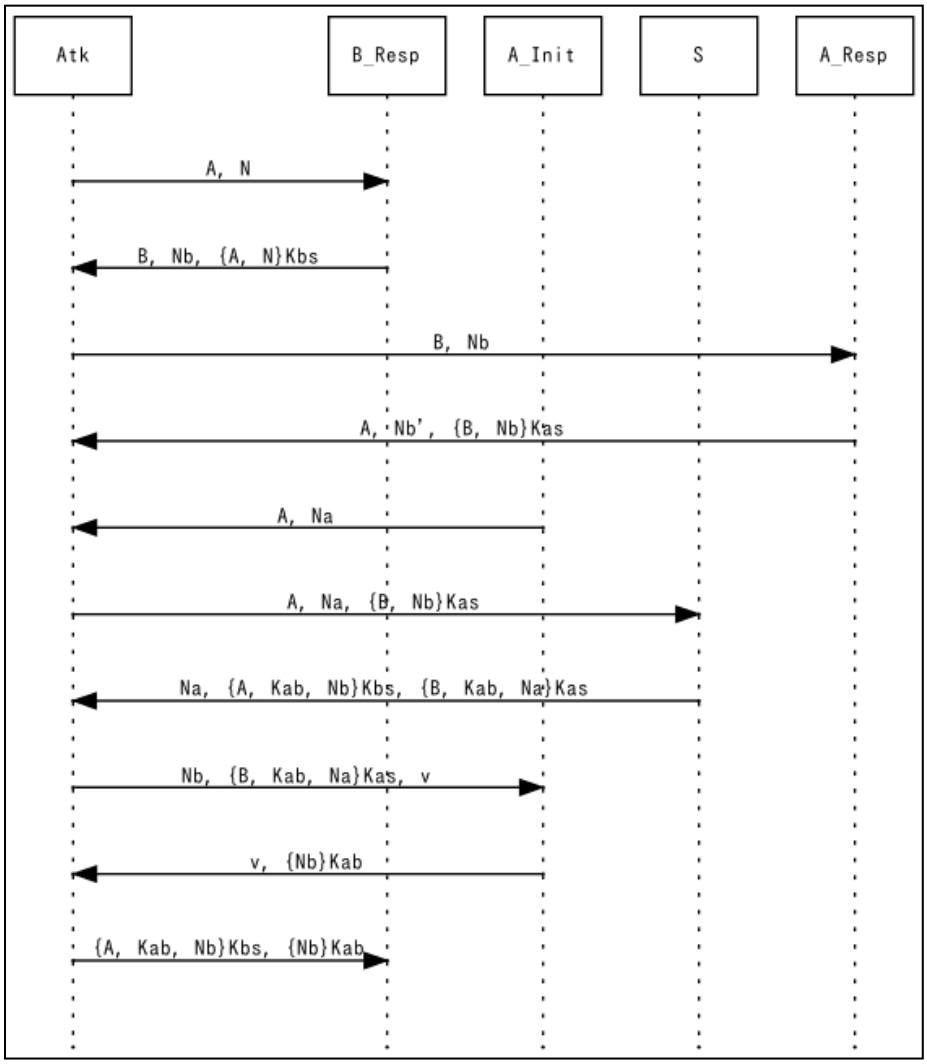

図 2.攻撃シーケンス

4. 形式化

4.1. 方針

SPORE ライブラリ（http://www.lsv.ens-cachan.fr/Software/spore/）の記述に従い形式化した。

4.2. 妥当性

特になし。

4.3. 検証ツールとの相性

プロトコル仕様、攻撃者モデル、セキュリティ要件を ProVerif で記述するにあたって、特に制限はなかった。

4.4. 検証ツール適用時の性能

検証時間は 0.8 秒であった。実行環境は以下のとおり。

- ✧ Intel Core i7 L620 2.00HGz
- ✧ Windows7 上の VirtualBox 仮想マシン上の Ubuntu Linux 12.04.1 LTS
- ✧ メモリ 512MB
- ✧ ProVerif 1.86pl3

5. 備考

本文書は、**総務省「暗号・認証技術等を用いた安全な通信環境推進事業に関する実証実験の請負　成果報告書**」からの引用である。