IKE-SIG 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

  The Internet Key Exchange (IKE)

- ✧ 機能

  IPsec の前に実行する鍵交換プロトコル。認証に電子署名を使用。

- ✧ 関連する標準

  RFC2409 (https://tools.ietf.org/html/rfc2409)

2. ProVerif の文法による記述

フェーズ 1 の署名による認証(Main モード)を記述・評価した。また、ヘッダ及びオプション扱いのペイロードは省略した。

2.1. プロトコル仕様

```
 (*
set ignoreTypes = attacker.
*)


free c: channel.


(*
type mackey.
*)




type host.
type pkey.
type skey.
type symkey.
```

```
type nonce.
type cookie.
type G.
type Z.

(* PKE *)

fun pk(skey): pkey.
fun encrypt(bitstring, pkey): bitstring.
reduc forall x: bitstring, y: skey; decrypt(encrypt(x, pk(y)), y) = x.



(* SIG *)

fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, k: skey; getmess(sign(m, k)) = m.
reduc forall m: bitstring, k: skey; checksign(sign(m, k), pk(k)) = m.

(* SKE *)

fun symencrypt(bitstring, bitstring): bitstring.
reduc forall x: bitstring, y: bitstring; symdecrypt(symencrypt(x, y),
y) = x.



(* prf *)
fun prf(bitstring, bitstring): bitstring.

(* DH *)
const g: G.
fun exp(G, Z): G.
equation forall x: Z, y: Z; exp(exp(g, x), y) = exp(exp(g, y), x).
```

```
fun G_to_bitstring(G): bitstring [typeConverter].

(* table *)
table keys(host, pkey).

(* constants *)
const MAIN, QUICK, AGGRESSIVE: bitstring.
const SIG, PKE, PRESHARED: bitstring.
const c0, c1, c2: bitstring.
const null: bitstring.

(* function for checking secrecy *)
fun hide(bitstring, bitstring): bitstring.
reduc forall x: bitstring, y: bitstring;
      reveal(hide(x, y), y) = x.

(* Constants for describing queries *)
free secretInit, secretResp, secretForTest: bitstring [private].

(* Honest hosts *)
free hostA, hostB, CA: host.

(* events *)
event initiatorKey1(host, bitstring, bitstring, bitstring, bitstring,
bitstring).
event initiatorKey2(host, host, bitstring, bitstring, bitstring,
bitstring, bitstring).
event responderKey1(host, host, bitstring, bitstring, bitstring,
bitstring, bitstring).
event responderKey2(host, host, bitstring, bitstring, bitstring,
bitstring, bitstring).
event end().
```

```
(* assumptions *)
not attacker(new skA).
not attacker(new skB).
not attacker(new skCA).

(* queries *)
(*
query x: host, y: host; inj-event(endBparam(x,y)) ==>
inj-event(beginBparam(x,y)).
*)
query attacker(secretInit); attacker(secretResp).

query
  idii: host, idir: host, sa: bitstring, skeyid: bitstring,
  skeyid_d: bitstring, skeyid_a: bitstring, skeyid_e: bitstring;
  inj-event(initiatorKey2(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e)) ==>
  inj-event(responderKey1(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e));
  inj-event(responderKey2(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e)) ==>
  inj-event(initiatorKey1(idii, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e)).

let processInitiatorMainSig(skCA: skey, skA: skey, skB: skey) =
    (* Parameters are given from the network *)
    in(c, (IDii: host, CERT_I: bitstring, SAi: bitstring));
    let m = MAIN in
    if IDii = hostA || IDii = hostB then
    let skii = (if IDii = hostA then skA else skB) in
    (* 1  Proposing SA *)
```

```
    new CKY_I: cookie;
    out(c, ((CKY_I, m), SAi));
    (* 1' Replay *)
    in(c, ((=CKY_I, CKY_R: cookie, =m), SAr: bitstring));
    if SAr = SAi then
    (* 2  (p.24) *)
    new xi: Z;
    let gxi = exp(g, xi) in
    new Ni: nonce;
    let HDR = (CKY_I, CKY_R, m) in
    out(c, (HDR, gxi, Ni));
    (* 2' (p.24) *)
    in(c, (=HDR, gxr:G, Nr: nonce));
    (* 3 (p.25)*)
    let gxy = exp(gxr, xi) in
    let SKEYID = prf((Ni, Nr), G_to_bitstring(gxy)) in
    let SKEYID_d = prf(SKEYID, (gxy, CKY_I, CKY_R, c0)) in
    let SKEYID_a = prf(SKEYID, (SKEYID_d, gxy, CKY_I, CKY_R, c1)) in
    let SKEYID_e = prf(SKEYID, (SKEYID_a, gxy, CKY_I, CKY_R, c2)) in
    let HASH_I = prf(SKEYID, (gxi, gxr, CKY_I, CKY_R, SAi, IDii)) in
    event initiatorKey1(IDii, SAi, SKEYID, SKEYID_d, SKEYID_a,
SKEYID_e);
    out(c, (HDR, symencrypt((IDii, CERT_I, sign(HASH_I, skii)),
SKEYID_e)));
    (* 3' (p.25) *)
    in(c, (=HDR, ct: bitstring));
    let (IDir: host, CERT_R: bitstring, SIG_R: bitstring)
         = symdecrypt(ct, SKEYID_e) in
    get keys(=IDir, pkir) in
    let HASH_R = prf(SKEYID, (gxr, gxi, CKY_R, CKY_I, SAi, IDir)) in
    if HASH_R = checksign(SIG_R, pkir) then
    (* events *)
```

```
    if IDir = hostA || IDir = hostB then
    if IDii <> IDir then
    out(c, (hide(secretInit, SKEYID),
            hide(secretInit, SKEYID_d),
            hide(secretInit, SKEYID_a),
            hide(secretInit, SKEYID_e)));
    event initiatorKey2(IDii, IDir, SAi, SKEYID, SKEYID_d, SKEYID_a,
SKEYID_e).

let processResponderMainSig(skCA: skey, skA: skey, skB: skey) =
    (* Parameters are given from the network *)
    in(c, (IDir: host, CERT_R: bitstring, SAr: bitstring));
    if IDir = hostA || IDir = hostB then
    let skir = (if IDir = hostA then skA else skB) in
    (* 1 *)
    in(c, (CKY_I: bitstring, mode: bitstring, SAi: bitstring));
    if SAi = SAr && mode = MAIN then
    (* 1' *)
    new CKY_R: bitstring;
    let HDR = (CKY_I, CKY_R, mode) in
    out(c, (HDR, SAr));
    (* 2 *)
    in(c, (=HDR, gxi: G, Ni: nonce));
    (* 2' *)
    new xr: Z;
    let gxr = exp(g, xr) in
    new Nr: nonce;
    out(c, (HDR, gxr, Nr));
    (* 3 *)
    let gxy = exp(gxi, xr) in
    let SKEYID = prf((Ni, Nr), G_to_bitstring(gxy)) in
    let SKEYID_d = prf(SKEYID, (gxy, CKY_I, CKY_R, c0)) in
```

```
    let SKEYID_a = prf(SKEYID, (SKEYID_d, gxy, CKY_I, CKY_R, c1)) in
    let SKEYID_e = prf(SKEYID, (SKEYID_a, gxy, CKY_I, CKY_R, c2)) in
    in(c, (=HDR, ct: bitstring));
    let HASH_R = prf(SKEYID, (gxr, gxi, CKY_R, CKY_I, SAi, IDir)) in
    let (IDii:host, CERT_I: bitstring, SIG_I: bitstring)
         = symdecrypt(ct, SKEYID_e) in
    let HASH_I = prf(SKEYID, (gxi, gxr, CKY_I, CKY_R, SAi, IDii)) in
    get keys(=IDii, pkii) in
    if HASH_I = checksign(SIG_I, pkii) then
    (* 3' *)
    event responderKey1(IDii, IDir, SAi, SKEYID, SKEYID_d, SKEYID_a,
SKEYID_e);
    out(c, (HDR, symencrypt((IDir, CERT_R, sign(HASH_R, skir)),
SKEYID_e)));
    (* events *)
    if IDii = hostA || IDii = hostB then
    out(c, (hide(secretResp, SKEYID),
            hide(secretResp, SKEYID_d),
            hide(secretResp, SKEYID_a),
            hide(secretResp, SKEYID_e)));
    event responderKey2(IDii, IDir, SAi, SKEYID, SKEYID_d, SKEYID_a,
SKEYID_e).

let keyRegistration =
    in(c, (h:host, k:pkey));
    if h <> hostA && h<> hostB then
        insert keys(h, k).

let CA(skCA: skey) =
    in(c, (h:host, k:pkey));
    if h <> hostA && h<> hostB && h<> CA then
       out(c, sign((h, k), skCA)).
```

```
process
          (* CA *)
          new skCA: skey;
          out(c, pk(skCA));
          (* hostA *)
          new skA: skey;
          insert keys(hostA, pk(skA));
         let CERT_A = sign((hostA, pk(skA)), skCA) in
          out(c, CERT_A);
          (* hostB *)
          new skB: skey;
          insert keys(hostB, pk(skB));
         let CERT_B = sign((hostB, pk(skB)), skCA) in
          out(c, CERT_B);
          ((!processInitiatorMainSig(skCA, skA, skB))|
           (!processResponderMainSig(skCA, skA, skB))|
           (!CA(skCA))|
           (!keyRegistration))
```

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 attacker(secretInit); attacker(secretResp).
query
  idii: host, idir: host, sa: bitstring, skeyid: bitstring,
  skeyid_d: bitstring, skeyid_a: bitstring, skeyid_e: bitstring;
  (* (2) *)
  inj-event(initiatorKey2(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e)) ==>
  inj-event(responderKey1(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
```

```
skeyid_e));
  (* (3) *)
  inj-event(responderKey2(idii, idir, sa, skeyid, skeyid_d, skeyid_a,
skeyid_e)) ==>
  inj-event(initiatorKey1(idii,  sa,  skeyid,  skeyid_d,  skeyid_a,
skeyid_e)).
```

(1) 交換した鍵の秘匿性。

(2) イニシエータによるレスポンダの認証。(互いの ID とセキュリティアソシエーションと交換した鍵の一致を含む。)

(3) レスポンダによるイニシエータの認証。(イニシエータの ID とセキュリティアソシエーションと交換した鍵の一致を含む。)

なお、レスポンダの ID の一致を(3)に含めたものは明らかに成り立たないため、これを省略した。

3. ProVerif による評価結果

3.1. 出力

安全性はいずれも成り立つこと (true) が確認された。本評価ではイニシエータが自分自身と暗号プロトコルを実行しないと仮定して評価を行なった。この仮定が正しくない場合は安全性(2)が成り立たたない。

```
RESULT      inj-event(endAuthenticator(p_30,a_31,r_32,k_33))      ==>
inj-event(beginPeer(p_30,a_31,r_32,k_33)) is false.
RESULT  (even  event(endAuthenticator(p_638,a_639,r_640,k_641))  ==>
event(beginPeer(p_638,a_639,r_640,k_641)) is false.)
RESULT not attacker(secretAuthenticator[]) is false.
RESULT not attacker(secretPeer[]) is false.
```

3.2. 攻撃の解説

前述したように、攻撃は発見されなかった。

4. 形式化

4.1. 方針

指数演算の可換性 exp(exp(x, y), z) = exp(exp(x, z), y) を形式化に組込んだ場合に評価が停止しなくなるため、一般の元 x のかわりに群の生成元 g についてのみ可換性を形

式化した。すなわち、exp(exp(g, y), z) = exp(exp(g, z), y) として形式化した。

4.2. 妥当性

形式化の制限により一般の指数演算の可換性を利用した攻撃を考慮できないため、評価結果で安全とされた性質についても攻撃が存在する可能性がある。

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。

ただし、ProVerif ツールの等式推論能力により、本暗号プロトコルのような指数演算を用いる暗号プロトコルを扱えたが、前述のような制限があるため必ずしも相性がよいとはいえない。

4.4. 検証ツール適用時の性能

検証時間は 19.5 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。