IKEv2 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

  Internet Key Exchange Protocol Version 2 (IKEv2)

- ✧ 機能

  IPsec の前に実行する相互認証・鍵交換プロトコル。 IKEv1 との互換性は確保されていない。 暗号化方式として 3DES、ハッシュ関数として SHA-1 が規定されている。

- ✧ 関連する標準

  RFC5996 (https://tools.ietf.org/html/rfc5996)

2. ProVerif の文法による記述

IKE_SA_INIT による鍵交換と IKE_AUTH による認証のみを記述・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 attacker = passive.
*)
free c: channel.


type host.
type nonce.
type skey.
type pkey.
type SPI_t.
type SA_t.


table keys(host, skey, pkey).
```

```
(* SIG *)
fun sign(bitstring, skey): bitstring.
fun pk(skey): pkey.

reduc forall x: bitstring, k: skey;
      check(sign(x, k), pk(k)) = x.

(* MAC and symmetric-key encryption *)
type symkey.
fun mac(bitstring, symkey): bitstring.
reduc forall k: symkey, x: bitstring;
      checkmac(mac(x, k), k) = x.

fun senc(bitstring, symkey): bitstring.
reduc forall k: symkey, x: bitstring;
      sdec(senc(x, k), k) = x.

type number.
const zero: number.
fun s(number): number.                (* Successor *)

type spi.
const spi_zero: spi.

const null: bitstring.

type exchangeType.
const IKE_SA_INIT: exchangeType.
const IKE_AUTH: exchangeType.
const CREATE_CHILD_SA: exchangeType.
const INFORMATIONAL: exchangeType.
```

```
(* Diffie-Hellman *)
type G.
type exponent.
const g: G.
fun exp(G, exponent): G.
equation forall x: exponent, y: exponent; exp(exp(g, x), y) = exp(exp(g,
y), x).

(* PRF *)
fun G_to_bs(G): bitstring [data, typeConverter].
fun host_to_bs(host): bitstring [data, typeConverter].
fun prf(bitstring, bitstring): bitstring.
fun prf_sk_d(bitstring, bitstring): bitstring.
fun prf_sk_ai(bitstring, bitstring): symkey.
fun prf_sk_ar(bitstring, bitstring): symkey.
fun prf_sk_ei(bitstring, bitstring): symkey.
fun prf_sk_er(bitstring, bitstring): symkey.
fun prf_sk_pi(bitstring, bitstring): bitstring.
fun prf_sk_pr(bitstring, bitstring): bitstring.

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

(* misc *)
free secret_for_keys: bitstring [private].

(* events *)
event begin_init_IKE_AUTH(host, host, SA_t, bitstring, bitstring,
bitstring).
event begin_init_IKE_AUTH_weak(host, bitstring).
event begin_resp_IKE_AUTH(host, host, SA_t, bitstring, bitstring,
```

```
bitstring).
event end_init_IKE_AUTH(host, host, SA_t, bitstring, bitstring,
bitstring).
event end_resp_IKE_AUTH(host, host, SA_t, bitstring, bitstring,
bitstring).

(* queries *)

query attacker(secret_for_keys).
query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_init_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)).
query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_init_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)).

query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_init_IKE_AUTH_weak(h1, seed)).

let initiator =
    in(c, (SAi1: SA_t, SAi2: SA_t, TSi: bitstring, TSr: bitstring, IDi:
host, IDr: host));
    if IDi = hostA || IDi = hostB then
    (*
      IKE_SA_INIT
      1. I->R: HDR, SAi1, KEi, Ni
      2. R->I: HDR, SAr1, KEr, Nr, [CERTREQ]
    *)
```

```
    new SPIi: number;
    new ei: exponent;
    let KEi1 = exp(g, ei) in
    new Ni: nonce;
    let ike_sa_init_request
        = ((SPIi, spi_zero, IKE_SA_INIT, zero), SAi1, KEi1, Ni) in
    out(c, ike_sa_init_request);
    in(c, ike_sa_init_response: bitstring);
    let ((=SPIi, SPIr: SPI_t, =IKE_SA_INIT, =zero), SAr1: SA_t, KEr1:
G, Nr: nonce)
        = ike_sa_init_response in
    (* SKEYSEED = prf(Ni | Nr, g^ir)
       {SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr }
         = prf+(SKEYSEED, Ni | Nr | SPIi | SPIr)
    *)
    let SKEYSEED = prf((Ni, Nr), G_to_bs(exp(KEr1, ei))) in
    let SK_d = prf_sk_d(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ai = prf_sk_ai(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ar = prf_sk_ar(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ei = prf_sk_ei(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_er = prf_sk_er(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_pi = prf_sk_pi(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_pr = prf_sk_pr(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    (*
      IKE_AUTH
      3. I->R: HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi,
TSr}
      4. R->I: HDR, SK {IDr, [CERT,]                    AUTH, SAr2, TSi,
TSr}
    *)
    if SAi1 = SAr1 then
    get keys(=IDi, ski, pki) in
```

```
    let AUTHi = sign((ike_sa_init_request, Nr, prf(SK_pi,
host_to_bs(IDi))), ski) in
    event begin_init_IKE_AUTH_weak(IDi, SKEYSEED);
    event begin_init_IKE_AUTH(IDi, IDr, SAi2, TSi, TSr, SKEYSEED);
    out(c, mac(senc((IDi, AUTHi, SAi2, TSi, TSr), SK_ei), SK_ai));
    in(c, mct: bitstring);
    let (=IDr, AUTHr: bitstring, =SAi2, =TSi, =TSr)
        = sdec(checkmac(mct, SK_ar), SK_er) in
    get keys(=IDr, skr, pkr) in
    if (ike_sa_init_response, Ni, prf(SK_pr, host_to_bs(IDr))) =
check(AUTHr, pkr) then
    (* for specifying security  *)
    if IDr = hostA || IDr = hostB then
    out(c, (senc(secret_for_keys, SK_ai),
            senc(secret_for_keys, SK_ar),
            senc(secret_for_keys, SK_ei),
            senc(secret_for_keys, SK_er)));
    event end_init_IKE_AUTH(IDi, IDr, SAi2, TSi, TSr, SKEYSEED).

let responder =
    in(c, (SAr1: SA_t, SAr2: SA_t, IDi: host, IDr: host));
    if IDr = hostA || IDr = hostB then
    (*
      IKE_SA_INIT
      1. I->R: HDR, SAi1, KEi, Ni
      2. R->I: HDR, SAr1, KEr, Nr, [CERTREQ]
    *)
    in(c, ike_sa_init_request: bitstring);
    let ((SPIi: bitstring, =spi_zero, =IKE_SA_INIT, =zero),
         SAi1: SA_t, KEi: G, Ni: nonce) = ike_sa_init_request in
    new SPIr: number;
    new er: exponent;
```

```
    new Nr: nonce;
    let ike_sa_init_response
        = ((SPIi, SPIr, IKE_SA_INIT, zero), SAr1, exp(g, er), Nr) in
    let SKEYSEED = prf((Ni, Nr), G_to_bs(exp(KEi, er))) in
    let SK_d = prf_sk_d(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ai = prf_sk_ai(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ar = prf_sk_ar(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_ei = prf_sk_ei(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_er = prf_sk_er(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_pi = prf_sk_pi(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    let SK_pr = prf_sk_pr(SKEYSEED, (Ni, Nr, SPIi, SPIr)) in
    out(c, ike_sa_init_response);
    (*
      IKE_AUTH
      3. I->R: HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi,
TSr}
      4. R->I: HDR, SK {IDr, [CERT,]                    AUTH, SAr2, TSi,
TSr}
    *)
    in(c, mct2: bitstring);
    let (=IDi, AUTHi: bitstring, SAi2: SA_t, TSi: bitstring, TSr:
bitstring)
        = sdec(checkmac(mct2, SK_ai), SK_ei) in
    get keys(=IDi, ski, pki) in
    if (ike_sa_init_request, Nr, prf(SK_pi, host_to_bs(IDi))) =
check(AUTHi, pki) then
    get keys(=IDr, skr, pkr) in
    let AUTHr = sign((ike_sa_init_response, Ni, prf(SK_pr,
host_to_bs(IDr))), skr) in
    event begin_resp_IKE_AUTH(IDi, IDr, SAr2, TSi, TSr, SKEYSEED);
    out(c, mac(senc((IDr, AUTHr, SAr2, TSi, TSr), SK_er), SK_ar));
    (* for specifying security *)
```

```
    if IDi = hostA || IDi = hostB then
    out(c, (senc(secret_for_keys, SK_ai),
            senc(secret_for_keys, SK_ar),
            senc(secret_for_keys, SK_ei),
            senc(secret_for_keys, SK_er)));
    event end_resp_IKE_AUTH(IDi, IDr, SAr2, TSi, TSr, SKEYSEED).

let keyregistration =
    in(c, (h:host , ks:skey, kp:pkey));
    if h <> hostA && h <> hostB then
    insert keys(h, ks, kp).

process
         new skA: skey;
         insert keys(hostA, skA, pk(skA));
         out(c, pk(skA));
         new skB: skey;
         out(c, pk(skB));
         insert keys(hostB, skB, pk(skB));
         ((!initiator)|(!responder)|(!keyregistration)
          (* for debug *)
          (*
         |
          (new SA: SA_t; new TS: bitstring;
           (out(c, (SA, SA, TS, TS, hostA, hostB)) |
            out(c, (SA, SA, hostA, hostB))))
              *)
         )
```

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(secret_for_keys).
(* (2) *)
query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_init_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)).
(* (3) *)
query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_init_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)).
(* (4) *)
query h1: host, h2: host, sa: SA_t, tsi: bitstring, tsr: bitstring, seed:
bitstring;
      inj-event(end_resp_IKE_AUTH(h1, h2, sa, tsi, tsr, seed)) ==>
      inj-event(begin_init_IKE_AUTH_weak(h1, seed)).
```

(1) 交換された暗号鍵及び MAC 鍵の秘匿性。

(2) イニシエータによるレスポンダの強い認証。

(3) レスポンダによるイニシエータの強い認証。

(4) レスポンダによるイニシエータの弱い認証。

（イニシエータの名前と鍵生成のシードの一致。）

ここでは「強い認証」はレスポンダ及びレスポンダの ID、セキュリティアソシエーション、トラフィックセレクタ及び鍵生成のシードの一致をいう。

3. ProVerif による評価結果

3.1. 出力

安全性のうち(3)以外は成り立つこと（true）が分かった。(3)については攻撃が発見された。ただし、オプション扱いとなっているペイロードを送信することにすれば安全性が成り立つ可能性がある。

```
RESULT inj-event(end_resp_IKE_AUTH(h1,h2,sa,tsi,tsr,seed))
 ==> inj-event(begin_init_IKE_AUTH_weak(h1,seed)) is true.
RESULT inj-event(end_resp_IKE_AUTH(h1_144974,h2_144975,
sa_144976,tsi_144977,tsr_144978,seed_144979))
 ==> inj-event(begin_init_IKE_AUTH(h1_144974,h2_144975,sa_144976,
tsi_144977,tsr_144978,seed_144979)) is false.
RESULT (even event(end_resp_IKE_AUTH(h1_289681,h2_289682,
sa_289683,tsi_289684,tsr_289685,seed_289686))
 ==>event(begin_init_IKE_AUTH(h1_289681,h2_289682,
sa_289683,tsi_289684,tsr_289685,seed_289686)) is false.)
RESULT inj-event(end_init_IKE_AUTH(h1_290527,h2_290528,
sa_290529,tsi_290530,tsr_290531,seed_290532))
==>inj-event(begin_resp_IKE_AUTH(h1_290527,
h2_290528,sa_290529,tsi_290530,tsr_290531,seed_290532)) is true.
RESULT not attacker(secret_for_keys[]) is true.
```

3.2. 攻撃の解説

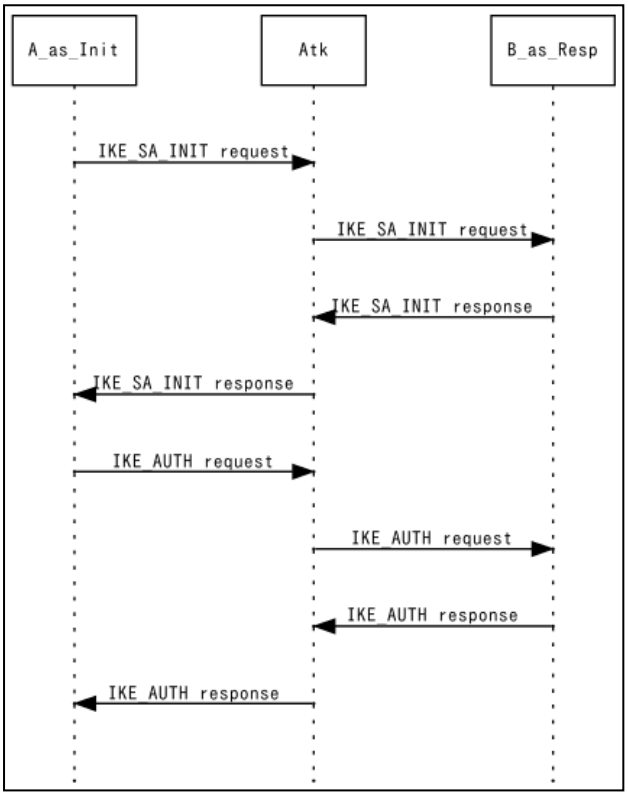

図 1　攻撃シーケンス

ホストAがイニシエータ(Initiator)の役割を攻撃者を相手として行い、ホストBがレスポンダ(Responder)の役割をホストAを相手として行う場合を考える。攻撃者は単にホストAとホストBが送信したメッセージを互いに転送するとする。このとき、ホストAが攻撃者を相手としているにもかかわらず、ホストBは暗号プロトコルを完了してしまう。さらに、このときホストAとホストBで異なるセキュリティアソシエーションを確立してしまう。

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 49.6 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。