PKMv2 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

  PKM (Privacy and Key Management Protocol Version 2)

- ✧ 機能

  Wimax 通信における端末 (SS/MS) と基地局 (BS) 間の認証・鍵交換プロトコル。

- ✧ 関連する標準

  IEEE Std 802.16e-2005

  (http://standards.ieee.org/getieee802/download/802.16e-2005.pdf)

2. ProVerif の文法による記述

PKM の phase1/phase2 を 1 つの暗号プロトコルとし、RSA 認証の場合を評価した。

2.1. プロトコル仕様

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


free c: channel.


type host.
type nonce.
type pkey.
type skey.
type symkey.
type mackey.


(* RSA-OAEP (?) encryption *)
```

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

(* Symmetric-Key encryption (AES?) *)
fun sencrypt(bitstring, symkey): bitstring.
reduc forall x: bitstring, k: symkey; sdecrypt(sencrypt(x, k), k) = x.

(* Key encryption (AES?) *)
fun kencrypt(symkey, symkey): bitstring.
reduc forall x: symkey, k: symkey; kdecrypt(kencrypt(x, k), k) = x.

(* RSA-SHA1 signature *)

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.

(* CMAC/HMAC *)
fun mac(bitstring, mackey): bitstring.
reduc forall m: bitstring, k: mackey; getmacmess(mac(m, k)) = m.
reduc forall m: bitstring, k: mackey; checkmac(mac(m, k), k) = m.

(* Derivation of AK from prePAK and
                  AKID, MAC_KEY_D, MAC_KEY_U, KEK from AK *)
fun d_ak(bitstring): bitstring.

fun d_akid(bitstring): bitstring.
fun d_mac_key_d(bitstring) : mackey.
fun d_mac_key_u(bitstring) : mackey.
fun d_kek(bitstring): symkey.
```

```
(* Type converter *)
fun pkey_to_bitstring(pkey): bitstring [data, typeConverter].
fun nonce_to_bitstring(nonce): bitstring [data, typeConverter].

(* Honest hosts *)
free OP, CA, honestM, honestSS, honestBS: host.

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

(* Events *)
event tek_issued(host, host, nonce, symkey).
event tek_accepted(host, host, nonce, symkey).
event rsa_ack_received(host, host, bitstring).
event rsa_ack_issued(host, host, bitstring).
event rsa_ack_issued_noBS(host, nonce, bitstring).
event end().  (* for testing *)

(*
event tek_request_accepted().
event bs_accept_authentication().
*)

(* Constants for describing queries *)
free secretSSTEK, secretBSTEK: symkey [private].
free secretForTest: bitstring [private].



(* Queries *)
```

```
(*
query attacker(secretForTest).
*)

  (* 1 *)
query x: host, y: host, n: nonce, k: symkey;
      inj-event(tek_accepted(x, y, n, k)) ==>
      inj-event(tek_issued(x, y, n, k)).
  (* 2 *)
query x: host, y: host, n: nonce, prePAK: bitstring;
      inj-event(rsa_ack_received(x, y, prePAK)) ==>
      inj-event(rsa_ack_issued(x, y, prePAK)).
  (* 3 *)
query attacker(secretSSTEK).
  (* 4 *)
query attacker(secretBSTEK).

(*
query x: host, y: host, n: nonce, prePAK: bitstring;
      inj-event(rsa_ack_received(x, y, n, prePAK)) ==>
      inj-event(rsa_ack_issued_noBS(y, n, prePAK)).
*)

(* Protocol *)
let processSS(skSS: skey, SSCert: bitstring, MCert: bitstring, pkOP:
pkey) =
    in(c, (Cap: bitstring, Cap2: bitstring, SecNegParam: bitstring,
            PKMConfSettings: bitstring, BasicCID: bitstring));
    (* Auth Info *)
    out(c, MCert);
    (* RSA Request *)
    new Ns: nonce;
```

```
    let rsa_req_text = (SSCert, Ns, Cap, BasicCID) in
    out(c, sign(rsa_req_text, skSS));
    (* RSA Reply *)
    in(c, rsa_rep: bitstring);
    let rsa_rep_text = getmess(rsa_rep) in
    let (=Ns, Nb: nonce, encrypted_prePAK: bitstring, KeyLifetime:
bitstring,
         SeqNumber: bitstring, BSCert: bitstring) = rsa_rep_text in
    let (BS: host, pkBS: pkey) = checksign(BSCert, pkOP) in
    if rsa_rep_text = checksign(rsa_rep, pkBS) then
    let prePAK = decrypt(encrypted_prePAK, skSS) in
    (* RSA Ack *)
    event rsa_ack_issued(BS, honestSS, prePAK);
    event rsa_ack_issued_noBS(honestSS, Nb, prePAK);
    out(c, sign(nonce_to_bitstring(Nb), skSS));
    (* Deriving Keys *)
    let ak = d_ak(prePAK) in
    let akid = d_akid(ak) in
    let mac_key_d = d_mac_key_d(ak) in
    let mac_key_u = d_mac_key_u(ak) in
    let kek = d_kek(ak) in
    (* SA-TEK-Challenge *)
    in(c, sa_tek_challenge: bitstring);
    let (Nb2: nonce, =akid) = checkmac(sa_tek_challenge, mac_key_d) in
    (* SA-TEK-Request *)
    new Ns2: nonce;
    let sa_tek_req_text
         = (Ns2, Nb2, akid, Cap2, SecNegParam, PKMConfSettings) in
    out(c, mac(sa_tek_req_text, mac_key_u));
    (* SA-TEK-Response *)
    in(c, sa_tek_resp: bitstring);
    let (=Ns2, =Nb2, =akid, SAdesc: bitstring, secnegparam': bitstring)
```

```
=
        checkmac(sa_tek_resp, mac_key_d) in
    let SAID = SAdesc in
    (* Key Request *)
    new Ns3: nonce;
    let key_req_text = (SAID, Ns3) in
    out(c, mac(key_req_text, mac_key_u));
    (* Key Response *)
    in(c, key_resp: bitstring);
    let (=SAID, encrypted_tek: bitstring, =Ns3)
        = checkmac(key_resp, mac_key_d) in
    let tek = kdecrypt(encrypted_tek, kek) in
    (* Testing Security *)
    if BS = honestBS then
    event tek_accepted(BS, honestSS, Ns3, tek);
    out(c, kencrypt(secretSSTEK, tek));
    event end().

let processBS(skBS: skey, BSCert: bitstring, pkCA: pkey) =
    in(c, (KeyLifetime: bitstring, SeqNumber: bitstring, SAdesc:
bitstring,
           SecNegParam': bitstring));
    (* Auth Info *)
    in(c, MCert: bitstring);
               (* It is important to check honestM here *)
    let (=honestM, pkM: pkey) = checksign(MCert, pkCA) in
    (* RSA Request *)
    in(c, rsa_req: bitstring);
    let (SSCert: bitstring, Ns: nonce, Cap: bitstring, BasicCID:
bitstring)
         = getmess(rsa_req) in
    let (SS: host, pkSS: pkey) = checksign(SSCert, pkM) in
```

```
    let rsa_req_text = checksign(rsa_req, pkSS) in
    (* RSA Reply *)
    new Nb: nonce;
    new prePAK: bitstring;
    let rsa_rep_text
         = (Ns, Nb, encrypt(prePAK, pkSS), KeyLifetime, SeqNumber,
BSCert) in
    out(c, sign(rsa_rep_text, skBS));
    (* RSA Ack *)
    in(c, rsa_ack: bitstring);
    if nonce_to_bitstring(Nb) = checksign(rsa_ack, pkSS) then
    (* Deriving Keys *)
    let ak = d_ak(prePAK) in
    let akid = d_akid(ak) in
    let mac_key_d = d_mac_key_d(ak) in
    let mac_key_u = d_mac_key_u(ak) in
    let kek = d_kek(ak) in
    (* SA-TEK-Challenge *)
    new Nb2: nonce;
    let sa_tek_challenge_text = (Nb2, akid) in
    out(c, mac(sa_tek_challenge_text, mac_key_d));
    (* SA-TEK-Request *)
    in(c, sa_tek_request: bitstring);
    let (Ns2: nonce, =Nb2, =akid, Cap2: bitstring, SecNegParam:
bitstring,
         PKMConfSettings: bitstring) = checkmac(sa_tek_request,
mac_key_u) in
    (* SA-TEK-Response *)
    let sa_tek_resp_text = (Ns2, Nb2, akid, SAdesc, SecNegParam') in
    out(c, mac(sa_tek_resp_text, mac_key_d));
    (* Key Request *)
    in(c, key_req: bitstring);
```

```
    let (SAID: bitstring, Ns3: nonce) = checkmac(key_req, mac_key_u) in
    if SAID = SAdesc then (* !!!!!!!!!! *)
    (* Key Response *)
    new tek: symkey;
    let key_resp_text = (SAID, kencrypt(tek, kek), Ns3) in
    event tek_issued(honestBS, SS, Ns3, tek);
    out(c, mac(key_resp_text, mac_key_d));
    (* Testing Security *)
    if SS = honestSS then
    event rsa_ack_received(honestBS, SS, prePAK);
    out(c, kencrypt(secretBSTEK, tek));
    event end().

(* Key registration *)

let processCert(CAkey: skey) =
    in(c, (h: host, k: pkey));
    if h <> honestM && h <> honestSS && h <> honestBS then
    out(c, sign((h, k), CAkey)).

process
         (* Operator *)
         new skOP: skey;
         let pkOP = pk(skOP) in
         out(c, pkOP);
         (*
         insert keys(OP, pkOP);
         *)
         (* CA *)
         new skCA: skey;
         let pkCA = pk(skCA) in
         out(c, pkCA);
```

```
(*
insert keys(CA, pkCA);
*)
(* Manufacturer *)
new skM: skey;
let pkM = pk(skM) in
let MCert = sign((honestM, pkM), skCA) in
out(c, MCert);
(* SS *)
new skSS: skey;
let pkSS = pk(skSS) in
let SSCert = sign((honestSS, pkSS), skM) in
out(c, SSCert);
(* BS *)
new skBS: skey;
let pkBS = pk(skBS) in
let BSCert = sign((honestBS, pkBS), skOP) in
out(c, BSCert);
(* Corrupted SS *)
(*
new skCSS: skey;
new CSS: host;
let pkCSS = pk(skCSS) in
let CSSCert = sign((CSS, pkCSS), skM) in
out(c, (CSSCert, skCSS));
*)
(* Corrupted BS *)
(*
new CBS: host;
new skCBS: skey;
let pkCBS = pk(skBS) in
let CBSCert = sign((CBS, pkCBS), skOP) in
```

```
        out(c, (CBSCert, skCBS));
        *)
        (
                (!processSS(skSS, SSCert, MCert, pkOP)) |
                (!processBS(skBS, BSCert, pkCA)) |

                (!processCert(skOP)) |
                (!processCert(skCA)) |
                (!processCert(skM)) |
            0
        )
```

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 x: host, y: host, n: nonce, k: symkey;
      inj-event(tek_accepted(x, y, n, k)) ==>
      inj-event(tek_issued(x, y, n, k)).
  (* (2) *)
query x: host, y: host, n: nonce, prePAK: bitstring;
      inj-event(rsa_ack_received(x, y, prePAK)) ==>
      inj-event(rsa_ack_issued(x, y, prePAK)).
  (* (3) *)
query attacker(secretSSTEK).
  (* (4) *)
query attacker(secretBSTEK).
```

(1) ロール SS(端末)によるロール BS(基地局)の認証 (鍵 tek の一致)。

(2) ロール BS によるロール SS の認証 (鍵 prePAK の一致)。

(3) ロール SS が交換した鍵 tek の秘匿性。

(4) ロール BS が交換した鍵 tek の秘匿性。

3. ProVerif による評価結果

3.1. 出力

4 つのセキュリティ要件のうち、(2)以外は成り立つ（true）ことを確認できた。

```
RESULT not attacker(secretBSTEK[]) is true.
RESULT not attacker(secretSSTEK[]) is false.
RESULT    inj-event(tek_issued2(x_8240,y_8241,ak_8242,k_8243))    ==>
inj-event(ak_accepted(x_8240,y_8241,ak_8242)) is true.
RESULT      inj-event(tek_accepted(x_13543,y_13544,k_13546))      ==>
inj-event(tek_issued(x_13543,y_13544,ak_13545,k_13546)) is false.
RESULT    (even    event(tek_accepted(x_17716,y_17717,k_17719))    ==>
event(tek_issued(x_17716,y_17717,ak_17718,k_17719)) is false.)
```

3.2. 攻撃の解説

攻撃シーケンスを図 1 に示す。ロール SS は phase1 の最後のメッセージ(RSA Ack)でノンス Nb のみに署名したものを送信し、鍵 prePAK などの情報が含まれない。このため、ロール SS が攻撃者(Atk_as_BS)と通信を行い、正規のロール BS が攻撃者を相手に通信を行うとき、中間者攻撃が可能である。具体的には、正規のロール BS は自分が送ったノンス Nb への署名を受信するため、ロール SS が自分を相手に通信をしていると考えるが、実際にはノンス Nb を利用した攻撃者を相手に通信を行なっている。

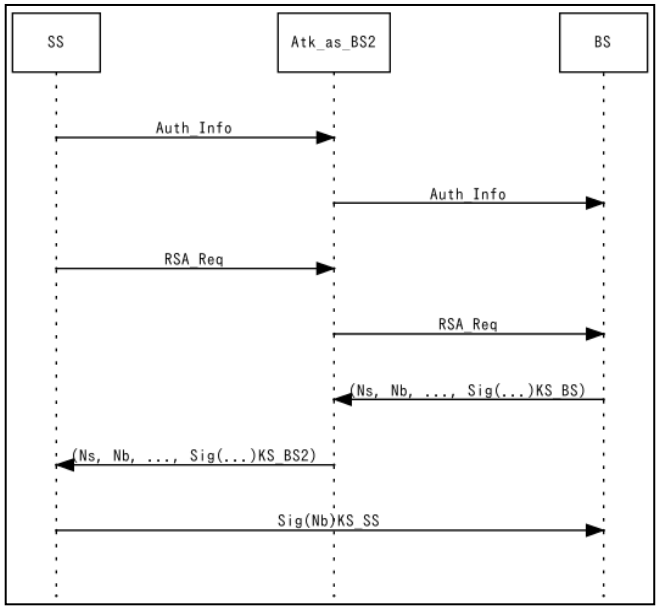

図 1.　攻撃シーケンス

4. 形式化

4.1. 方針

ProVerif の通常の形式化では評価が停止しなかったため、通常とは異なる形式化(set ignoreTypes = attacker.)を用いて評価を行なった。通常の形式化では正規参加者はメッセージの型チェックを行なわないが、このモデルでは正規参加者が型チェックを行う。

4.2. 妥当性

正規参加者が型チェックを行なう形式化を用いたため、正規参加者による型の混同を利用した攻撃が仮に存在したとしても本評価ではそれを発見できない。この点で、評価が不十分である可能性がある。

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。

4.4. 検証ツール適用時の性能

検証時間は 11.9 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。