PKM 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)

- ✧ 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 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_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, bitstring, symkey).
event tek_issued2(host, host, bitstring, symkey).
event tek_accepted(host, host, symkey).
event ak_accepted(host, host, 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, ak: bitstring, k: symkey;
      inj-event(tek_accepted(x, y, k)) ==>
      inj-event(tek_issued(x, y, ak, k)).

(* (2) *)
query x: host, y: host, ak: bitstring, k: symkey;
      inj-event(tek_issued2(x, y, ak, k)) ==>
      inj-event(ak_accepted(x, y, ak)).

(* (3) *)
query attacker(secretSSTEK).

(* (4) *)
query attacker(secretBSTEK).

(* Protocol *)
let processSS(skSS: skey, SSCert: bitstring, MCert: bitstring, pkOP:
pkey) =
    in(c, (BS: host, Cap: bitstring, Cap2: bitstring, SecNegParam:
bitstring,
            PKMConfSettings: bitstring, BasicCID: bitstring));
    (* Auth Info *)
    out(c, MCert);
    (* Auth Request *)
    let auth_req_text = (SSCert, Cap, BasicCID) in
    out(c, sign(auth_req_text, skSS));
    (* Auth Reply *)
    in(c, auth_rep: bitstring);
    let (encrypted_AK: bitstring, KeyLifetime: bitstring,
         SeqNumber: bitstring, SAdesc: bitstring) = auth_rep in
    (* Deriving Keys *)
    let ak = decrypt(encrypted_AK, skSS) 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
    (* for security *)
    event ak_accepted(BS, honestSS, ak);
    (* Key Request *)
    let SAID = SAdesc in
    let key_req_text = (SeqNumber, SAID) in
    out(c, mac(key_req_text, mac_key_u));
    (* Key Response *)
    in(c, key_resp: bitstring);
    let (=SeqNumber, =SAID, encrypted_tek: bitstring)
        = 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, 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
    (* Auth Request *)
    in(c, auth_req: bitstring);
    let (SSCert: bitstring, BasicCID: bitstring)
         = getmess(auth_req) in
```

```
    let (SS: host, pkSS: pkey) = checksign(SSCert, pkM) in
    let auth_req_text = checksign(auth_req, pkSS) in
    (* Auth Reply *)
    new ak: bitstring;
    let auth_rep
         = (encrypt(ak, pkSS), KeyLifetime, SeqNumber, SAdesc) in
    out(c, auth_rep);
    (* Deriving Keys *)
    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
    (* Key Request *)
    in(c, key_req: bitstring);
    let (=SeqNumber, SAID: bitstring) = checkmac(key_req, mac_key_u) in
    if SAID = SAdesc then (* !!!!!!!!!!! *)
    (* Key Response *)
    new tek: symkey;
    let key_resp_text = (SeqNumber, SAID, kencrypt(tek, kek)) in
    event tek_issued(honestBS, SS, ak, tek);
    out(c, mac(key_resp_text, mac_key_d));
    (* Testing Security *)
    if SS = honestSS then
    event tek_issued2(honestBS, SS, ak, tek);
    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, ak: bitstring, k: symkey;
```

```
      inj-event(tek_accepted(x, y, k)) ==>
      inj-event(tek_issued(x, y, ak, k)).

(* (2) *)
query x: host, y: host, ak: bitstring, k: symkey;
      inj-event(tek_issued2(x, y, ak, k)) ==>
      inj-event(ak_accepted(x, y, ak)).

(* (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 ak の一致)。

(3) ロール SS が交換した鍵の秘匿性。

(4) ロール BS が交換した鍵の秘匿性。

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)及び(4)は成り立つ（true）が、(1)及び(3)は成り立たない（false）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

ロール SS が送信するメッセージはロール BS の公開鍵による暗号化を行わず、ロール BS

から受けとるメッセージにはロール BS の秘密鍵による署名も行われないため、ロール SS は相手が攻撃者であっても鍵交換を完了してしまう。このため、攻撃者は鍵のシード ak を自由に決めることができ、鍵のシード ak から生成される MAC 鍵を知ることができる。これを利用して、攻撃者は自分が決めた鍵 tek に MAC をつけてロール SS に送信することで、ロール BS のふりをしつつ、ロール SS が使う鍵 tek を知ることができる。

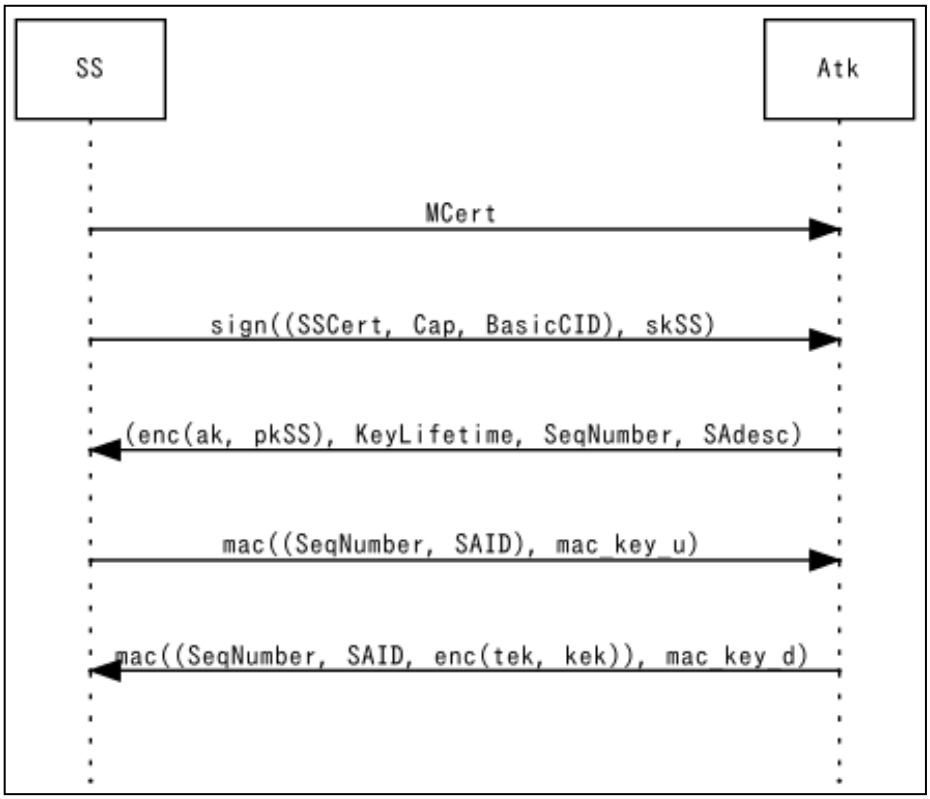

図 1.　攻撃シーケンス

4. 形式化

4.1. 方針

特になし。

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 0.1 秒未満であった。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。