EAP-AKA 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

  Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement

- ✧ 機能

  移動体通信（3G）における認証モジュール IM を用いた相互認証・鍵交換プロトコル。暗号として IM に組み込まれた AKA アルゴリズムを利用する。

- ✧ 関連する標準

  RFC4187 (http://www.ietf.org/rfc/rfc4187.txt)

2. ProVerif の文法による記述

CPVP 技術文書「**EAP-AKA の概要**」で示したロール S(Authenticator)をロール A で表す。

2.1. プロトコル仕様

```
free c: channel.


type key.
type nonce.
type host.


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


(* Definitions of hash function and mac.
   To allow more attacks, we assume all hash functions to be same
   and all macs to be same.
 *)
fun h(bitstring): bitstring.
```

```
fun hmac(key, bitstring): bitstring.
reduc forall x: bitstring; SHA1(x) = h(x).
reduc forall x: bitstring; FIPS_PRF(x) = bitstring_to_key(h(x)).
reduc forall k: key, x: bitstring; HMAC_SHA1(k, x) = hmac(k, x).
reduc forall k: key, r: nonce; f1(k, r) = hmac(k, nonce_to_bitstring(r)).
reduc forall k: key, r: nonce; f2(k, r) = hmac(k, nonce_to_bitstring(r)).
reduc forall k: key, r: nonce; f3(k, r) = hmac(k, nonce_to_bitstring(r)).
reduc forall k: key, r: nonce; f4(k, r) = hmac(k, nonce_to_bitstring(r)).
reduc forall k: key, r: nonce; f5(k, r) = hmac(k, nonce_to_bitstring(r)).

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

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

(*
fun SHA1(bitstring): bitstring.
fun FIPS_PRF(bitstring): key.
fun HMAC_SHA1(key, bitstring): bitstring.
fun f1(key, nonce): bitstring.
fun f2(key, nonce): bitstring.
fun f3(key, nonce): bitstring.
fun f4(key, nonce): bitstring.
fun f5(key, nonce): bitstring.
*)

const ReqId: bitstring [data].
const ResId: bitstring [data].
const Success: bitstring [data].
```

```
free hostA, hostP: host.

(* events *)
event endAuthenticator(host, host, nonce, key).
event beginPeer(host, host, nonce, key).

free secretAuthenticator, secretPeer: bitstring [private].

(*queries*)
query attacker(secretAuthenticator);
      attacker(secretPeer).

query p: host, a: host, r: nonce, k: key;
      inj-event(endAuthenticator(p, a, r, k)) ==>
      inj-event(beginPeer(p, a, r, k)).

let procAuthenticator(a: host) =
    out(c, ReqId);
    in(c, (=ResId, p: host));
    get keys(=p, =hostA, k) in
    new at_rand: nonce;
    let ck = f3(k, at_rand) in
    let ik = f4(k, at_rand) in
    let ak = f5(k, at_rand) in
    let at_autn = f1(k, at_rand) in
    let mk = SHA1((p, ik, ck)) in
    let k_aut = FIPS_PRF(mk) in
    let at_mac = HMAC_SHA1(k_aut, (at_rand, at_autn)) in
    out(c, (at_rand, at_autn, at_mac));
    in(c, (at_res: bitstring, at_mac2: bitstring));
    if at_mac2 = HMAC_SHA1(k_aut, at_res) &&
       at_res = f2(k, at_rand) then
```

```
    out(c, Success);
    if p = hostP then
    (* 以下の箇所を有効にする安全になる *)
    (*
    if at_autn <> at_res then
    *)
    out(c, encrypt(secretAuthenticator, k_aut));
    event endAuthenticator(p, a, at_rand, k_aut).

let procPeer(p: host) =
    in(c, a: host);
    get keys(=p, =a, k) in
    (**)
    in(c, =ReqId);
    out(c, (ResId, p));
    in(c, (at_rand: nonce, at_autn: bitstring, at_mac: bitstring));
    let ck = f3(k, at_rand) in
    let ik = f4(k, at_rand) in
    let ak = f5(k, at_rand) in
    let mk = SHA1((p, ik, ck)) in
    let k_aut = FIPS_PRF(mk) in
    if at_mac = HMAC_SHA1(k_aut, (at_rand, at_autn)) &&
       at_autn = f1(k, at_rand) then
    let at_res = f2(k, at_rand) in
    let at_mac2 = HMAC_SHA1(k_aut, at_res) in
    event beginPeer(p, a, at_rand, k_aut);
    out(c, (at_res, at_mac2));
    in(c, =Success);
    (* 以下の箇所を有効にする安全になる *)
    (*
    if at_autn <> at_res then
    *)
```

```
    if a = hostA then
    out(c, encrypt(secretPeer, k_aut)).

let keyRegistration =
    in(c, (h1: host, h2: host, k: key));
    if (h1, h2) <> (hostP, hostA) then
    insert keys(h1, h2, k).

process
         new Kpa: key;
         insert keys(hostP, hostA, Kpa);
         ((!procPeer(hostP))|
          (!procAuthenticator(hostA))|
          (!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 p: host, a: host, r: nonce, k: key;
      inj-event(endAuthenticator(p, a, r, k)) ==>
      inj-event(beginPeer(p, a, r, k)).
(* (2) *)
query attacker(secretAuthenticator);
      attacker(secretPeer).
```

(1)ロール A(Authenticator)によるロール P(Peer)の認証及び交換したノンスと鍵の一致。

(2)交換した鍵の秘匿性。

3. ProVerif による評価結果

3.1. 出力

安全性はともに成り立たない (false) という結果が出力された。

```
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

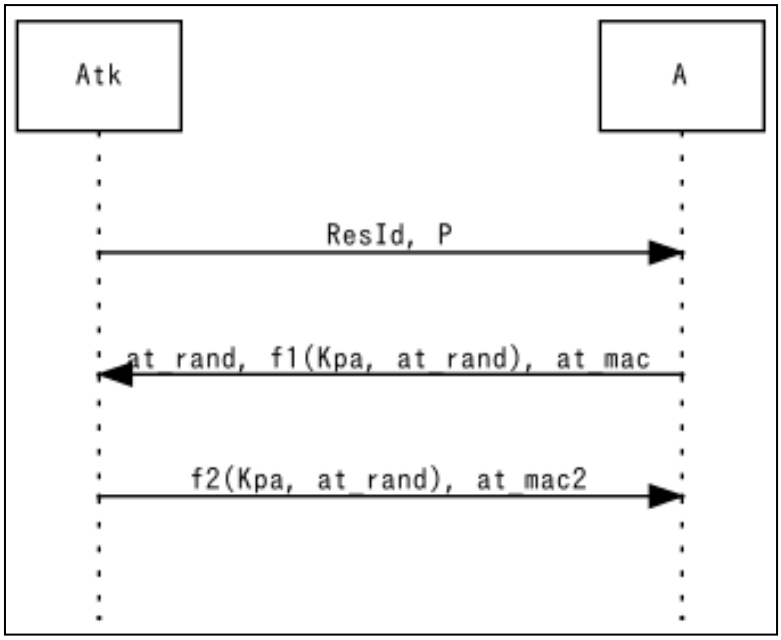

図 3.2.1　攻撃シーケンス

攻撃者が関数 f1(Kpa, at_rand)から関数 f2(Kpa, at_rand), 関数 f3(Kpa, at_rand), 関数 f4(Kpa, at_rand)を計算できる場合に、攻撃者がロール P になりすますことが可能。このとき、攻撃者はロール A から受けとった関数 f1(Kpa, at_rand)から関数 f2(Kpa, at_rand)と mac2 を計算して送りかえす。mac2 を計算するたには鍵 k_aut が必要だが、これは関数 f3(k, at_rand)と関数 f4(k, at_rand)から計算できるため、攻撃者は正しい mac2 を計算できる。また、鍵 k_aut の秘匿性も成り立たない。

4. 形式化

4.1. 方針

暗号プロトコルではハッシュ関数あるいは MAC として 2 引数の関数 f1, f2, f3, f4, f5

が用いられる。これらの関数は、オペレーター(携帯電話会社)が実装することになっており、必要な安全性要件は明確には定義されていない。このため、これらの関数は理想的な一方向性関数であるとした。その一方、これらの関数は 1 回の暗号プロトコル実行では全て同じ鍵を第一引数として用いられるため、実装方法によってはたとえば関数 f1(k, x)から関数 f2(k, x)を知ることができる可能性がある。このため、できるだけ広範な攻撃を想定するため、関数 f1 から関数 f5 までは全て同じ関数であるとした。

4.2. 妥当性

関数 f1 から関数 f5 までは鍵つきのハッシュ関数あるいは MAC であるため、理想的な一方向性関数とみなしても問題ないと考えられる。また、関数 f1 から関数 f5 までを同じ関数とするのは、より安全側に倒した定式化であるため、健全な定式化になっているが、発見した攻撃が実際には不可能である可能性もある。

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 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。