Fujioka-Suzuki-Xagawa-Yoneyama 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

  Fujioka-Suzuki-Xagawa-Yoneyama (FSXY)

- ✧ 機能

  暗号として鍵カプセル化メカニズム (Key Encapsulation Mechanism, KEM) を用いた認証付き鍵交換プロトコルの一般的構成法。

- ✧ 関連する文書

  A.Fujioka, K.Suzuki, K.Xagawa, and K.Yoneyama, "Strongly secure authenticated key exchange from factoring, codes, and lattices," Cryptology ePrint Archive: Report 2012/211, http://eprint.iacr.org/2012/211.pdf.

2. ProVerif の文法による記述

以下では、エンティティ U_A 及びエンティティ U_B をそれぞれイニシエータ (Initiator) 及びレスポンダ (Responder) と呼ぶ。

2.1. プロトコル仕様

```
(*
IND-CCA KEM (KeyGen, EnCap, DeCap)と IND-CPA KEM (wKeyGen, wEnCap, wDeCap)  を仮定して安全性がいえるが、ProVerif では暗号は皆 IND-CCA になってしまう.
*)


(*
F, F', G は PRF だが、モデルとしては RO と同じになってしまっている
*)
```

```
(* bitwise の exclusive-OR のモデル
   この asbtraction だと、u, v が既知のとき、xor2(u||u', v'||v)から
   u', v'をとりだせない
   randomness が一様であることと同じ?
   randomness が一様な場合は computationally sound な検証ができる?
*)

(*
set maxHyp = 10.
set selFun = Nounifset.
set maxDepth = 10.
set verboseRules = true.
*)

type RSE.

fun xor2(RSE, RSE): RSE.
reduc forall x:RSE, y:RSE; decompose_xor2_1(xor2(x, y), x) = y.
reduc forall x:RSE, y:RSE; decompose_xor2_2(xor2(x, y), y) = x.
fun xor3(RSE, RSE, RSE): RSE.
reduc forall x:RSE, y:RSE, z:RSE;
      decompose_xor3_1(xor3(x, y, z), x) = xor2(y, z).
reduc forall x:RSE, y:RSE, z:RSE;
      decompose_xor3_2(xor3(x, y, z), y) = xor2(x, z).
reduc forall x:RSE, y:RSE, z:RSE;
      decompose_xor3_3(xor3(x, y, z), z) = xor2(x, y).

(* 疑似乱数関数 *)

type FS.
type outcome.
```

```
fun F(FS, FS): RSE.
fun F'(FS, FS): RSE.
fun G(FS, bitstring): RSE. (* ここはbitstringからRSEに変更した *)

(* KEM *)
type RSG.
type EK.  (* 論文にはない型 *)
type DK.  (* 論文にはない型 *)
type KS.
type CS.

fun Ek(RSG): EK.
fun Dk(RSG): DK.
fun EnCapK(EK, RSE): KS.
fun EnCapCT(EK, RSE): CS.
reduc forall rk: RSG, re: RSE;
      DeCap(Dk(rk), EnCapCT(Ek(rk), re)) = EnCapK(Ek(rk), re).

fun wEk(RSG): EK.
fun wDk(RSG): DK.
fun wEnCapK(EK, RSE): KS.
fun wEnCapCT(EK, RSE): CS.
reduc forall rk: RSG, re: RSE;
      wDeCap(wDk(rk), wEnCapCT(wEk(rk), re)) = wEnCapK(wEk(rk), re).

type SS.
fun Ext(SS, KS): FS.

free seed: SS.     (* seed for Ext*)

(* for key distribution *)
type HOST. (* 論文にはない型 *)
```

```
table keys(HOST, FS, EK, DK).

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

free c: channel.

(* for describing securecy *)
free secretInit: bitstring [private].
free secretResp: bitstring [private].
free publicMessage: bitstring.
free publicMessage2: bitstring.

fun encrypt(bitstring, RSE): bitstring.
reduc forall x: bitstring, k: RSE;
      decrypt(encrypt(x, k), k) = x.

event beginInit(HOST, HOST, RSE).
event beginResp(HOST, HOST, RSE).
event endInit(HOST, HOST, RSE).
event endResp(HOST, HOST, RSE).

(* for strong secrecy *)
fun fakeKey(HOST, HOST, bitstring, bitstring): RSE [private].

(* queries *)
query attacker(secretInit);
      attacker(secretResp).

query a: HOST, b: HOST, k: RSE;
      inj-event(endInit(a, b, k)) ==>
      inj-event(beginResp(a, b, k));
```

```
      inj-event(endResp(a, b, k)) ==>
      inj-event(beginInit(a, b, k)).

let init(hostA: HOST, hostB: HOST) =
    in(c, (A: HOST, B: HOST));
    if A = hostA || A = hostB then
    if A <> B then
    (* key exchange *)
    new rA1: FS; new rA1': FS; new rA2: RSG;     (* ephemeral *)
    get keys(=A, sA, ekA1, dkA1) in
    get keys(=B, sB, ekB1, dkB1) in
    let CTA1 = EnCapCT(ekB1, xor2(F(sA, rA1), F'(rA1', sA))) in
    let KA1  =  EnCapK(ekB1, xor2(F(sA, rA1), F'(rA1', sA))) in
    let ekA2 = wEk(rA2) in
    let dkA2 = wDk(rA2) in
    let Xa = (A, B, CTA1, ekA2) in
    out(c, Xa);
    in(c, Xb: bitstring);
    let (=A, =B, CTB1: CS, CTB2: CS) = Xb in
    let KB1 = DeCap(dkA1, CTB1) in
    let KB2 = wDeCap(dkA2, CTB2) in
    let K1' = Ext(seed, KA1) in
    let K2' = Ext(seed, KB1) in
    let K3' = Ext(seed, KB2) in
    let ST = (A, B, ekA1, ekB1, CTA1, ekA2, CTB1, CTB2) in
    let SK = xor3(G(K1', ST), G(K2', ST), G(K3', ST)) in
    (* additional message for strong authenticty guarantee *)
    event beginInit(A, B, SK);
    out(c, encrypt(publicMessage, SK));
    in(c, ct: bitstring);
    if publicMessage2 = decrypt(ct, SK) then
    (* for security *)
```

```
    if B = hostA || B = hostB then
       out(c, encrypt(secretInit, SK));
       event endInit(A, B, SK).

let resp(hostA: HOST, hostB: HOST) =
    in(c, B: HOST);
    if B = hostA || B = hostB then
    (* key exchange *)
    in(c, Xa: bitstring);
    let (A: HOST, =B, CTA1: CS, ekA2: EK) = Xa in
    if A <> B then
    new rB1: FS; new rB1': FS; new rB2: RSE;    (* ephemeral *)
    get keys(=A, sA, ekA1, dkA1) in
    get keys(=B, sB, ekB1, dkB1) in
    let CTB1 = EnCapCT(ekA1, xor2(F(sB, rB1), F'(rB1', sB))) in
    let KB1  =  EnCapK(ekA1, xor2(F(sB, rB1), F'(rB1', sB))) in
    let CTB2 = wEnCapCT(ekA2, rB2) in
    let KB2  =  wEnCapK(ekA2, rB2) in
    let Xb = (A, B, CTB1, CTB2) in
    out(c, Xb);
    let KA1 = DeCap(dkB1, CTA1) in
    let K1' = Ext(seed, KA1) in
    let K2' = Ext(seed, KB1) in
    let K3' = Ext(seed, KB2) in
    let ST = (A, B, ekA1, ekB1, CTA1, ekA2, CTB1, CTB2) in
    let SK = xor3(G(K1', ST), G(K2', ST), G(K3', ST)) in
    (* additional message for strong authenticty guarantee *)
    in(c, ct: bitstring);
    if publicMessage = decrypt(ct, SK) then
    event beginResp(A, B, SK);
    out(c, encrypt(publicMessage2, SK));
    (* Attacker's query *)
```

```
    if A = hostA || A = hostB then
       out(c, encrypt(secretResp, SK));
       event endResp(A, B, SK).

let key_registration =
    in(c, (h: HOST, s: FS, ek: EK, dk: DK));
    if h <> hostA && h <> hostB then
    insert keys(h, s, ek, dk).

process
         new sA: FS;
         new rkgA: RSG;
         insert keys(hostA, sA, Ek(rkgA), Dk(rkgA));
         out(c, Ek(rkgA));
         new sB: FS;
         new rkgB: RSG;
         insert keys(hostB, sB, Ek(rkgB), Dk(rkgB));
         out(c, Ek(rkgB));
         ((!init(hostA, hostB)) | (!resp(hostA, hostB)) |
(!key_registration))
```

強認証の評価のため、暗号プロトコルのシーケンスに加え、publicMessage 及び publicMessage2 を、互いに送りあうメッセージを追加する形で記述した。

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 a: HOST, b: HOST, k: RSE;
      (* 2 *)
```

```
    inj-event(endInit(a, b, k)) ==>
    inj-event(beginResp(a, b, k));
    (* 3 *)
    inj-event(endResp(a, b, k)) ==>
    inj-event(beginInit(a, b, k)).
```

提案されている暗号プロトコルは、鍵交換のために用いられる鍵が漏洩した場合でも交換した鍵の秘匿性を保証できるのが特徴である。しかし、ツールの制約のためここでは通常の認証鍵交換の安全性を記述・評価する。

(1) 交換した鍵の秘匿性。

(2) レスポンダによる強い認証(鍵の一致を含む)。

(3) イニシエータによる強い認証(鍵の一致を含む)。

3. ProVerif による評価結果

3.1. 出力

いずれのセキュリティ要件も成り立つ（true）ことを確認できた。

```
RESULT inj-event(endInit(a,b,k_92)) ==> inj-event(beginResp(a,b,k_92))
is true.
RESULT inj-event(endResp(a,b,k_92)) ==> inj-event(beginInit(a,b,k_92))
is true.
RESULT not attacker(secretInit[]) is true.
RESULT not attacker(secretResp[]) is true.
```

3.2. 攻撃の解説

前述のとおり、攻撃は発見されなかった 。

4. 形式化

4.1. 方針

いわゆる Dolev-Yao モデルによって形式化した。本プロトコルではビットごとの排他的論理和(exclusive-or)の演算が用いられる。この演算には結合則及び可換性が成り立つが、これをそのままツールの等式として定式化すると、検証が停止しないという問題があった。このため、排他的論理和についての等式を制限した。

4.2. 妥当性

排他的論理和の等式を制限したため、必ずしも妥当でない可能性がある。

4.3. 検証ツールとの相性

プロトコル仕様､攻撃者モデルを ProVerif で記述するにあたって､特に制限はなかった。

セキュリティ要件については、本プロトコルは中間鍵の漏洩を許したときのセッション鍵の秘匿性を保証するが、ProVerif にはこれをそのまま評価する機能はないため、通常の鍵交換の安全性を評価した。

4.4. 検証ツール適用時の性能

検証時間は 20.1 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。