プロトコルの形式的安全性検証ツール ProVerif

@tex2e

ProVerif 概要

- できること：
  プロトコルをモデル化したコードを記述
  → ProVerif ツールで実行
  → 脆弱性あり・なしの判定

- わかること：
  プロトコルのセキュリティ特性
  秘匿性、真正性、オフライン攻撃、前方秘匿性

- 今日のお話：
  サンプルプロトコルで検証
  - プロトコル α
  - プロトコル β

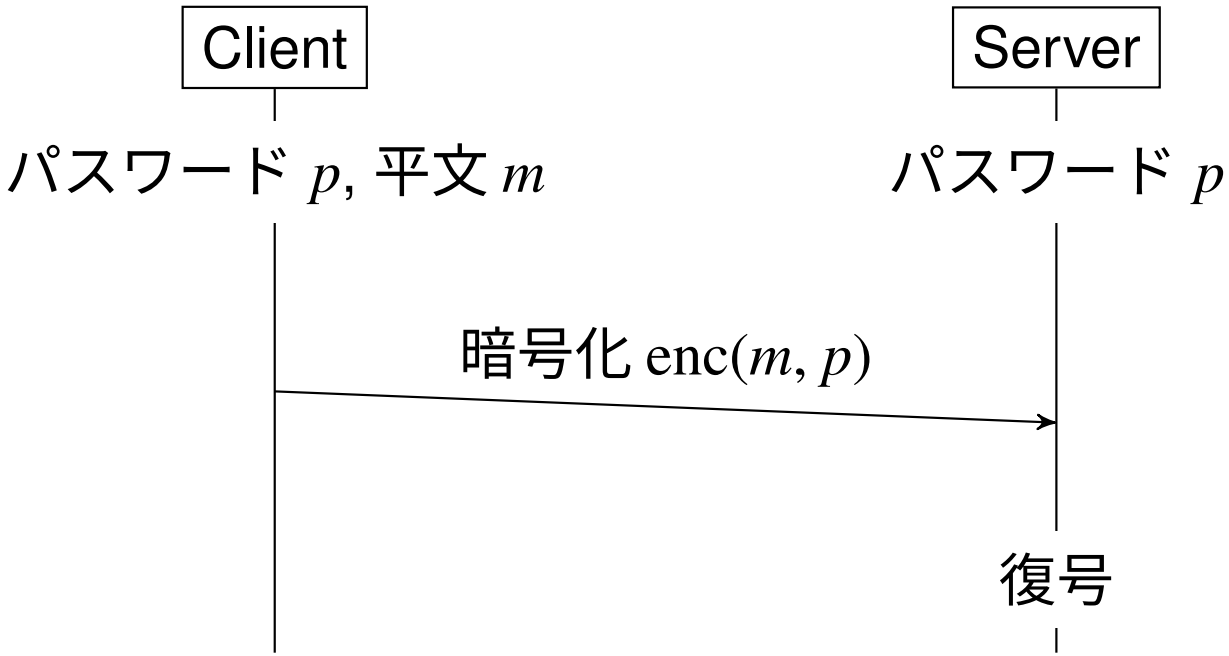
ProVerif

- プロトコルが安全かどうかを自動で証明
- 誰でも使えて無料 [1]
- spi 計算の書き方に (少し) 似ている
- 言語としては OCaml に (少し) 似ている

```
free c: channel.
free message: bitstring [private].

query attacker(message).

process
  out(c, message);
  0
```

[1] http://proverif.inria.fr/

プロトコル α



このプロトコルが安全かどうかを **ProVerif** で検証します

プロトコルのモデル化

ハッシュ関数

特徴：

- 一方向関数：$f(m) \rightarrow h$
- 完全な暗号モデル[2]において逆関数は存在しない

ProVerif のコード

```
fun hash(bitstring): bitstring.
```

[2] Dolve-Yao モデル

プロトコルのモデル化

共通鍵暗号

特徴：

- 暗号化：$\mathrm{enc}(m, k)$
- 復号　：$\mathrm{dec}(c, k)$
- 暗号化と復号で元に戻る：$\mathrm{dec}(\mathrm{enc}(m, k), k) = m$

ProVerif のコード

```
fun enc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key;
  dec(enc(m,k),k) = m.
```

プロトコル α のモデル化

クライアント–サーバ間の通信

```
(* クライアントA *)
let clientA() =
  event beginA(msg);
  out(c, enc(msg, password));
  0.

(* サーバB *)
let serverB() =
  in(c, x: bitstring);
  let recvmsg = dec(x, password) in
  event endB(recvmsg);
  0.

process
  ( (!clientA()) | (!serverB()) )
```

検証方法

秘匿性

限られた人しか情報にアクセスできないこと

- query ：検証クエリ
- attacker(v) ：攻撃者は変数 v に到達可能か

```
(* 秘匿性の検証 *)
query attacker(msg).
query attacker(password).
```

検証結果

秘匿性

```
$ proverif -color protocol1.pv
(
    {1}!
    {2}out(c, enc(msg,password))
) | (
    {3}!
    {4}in(c, x: bitstring);
    {5}let recvmsg: bitstring = dec(x,password) in
    0
)
...
--------------------------------------------------------------
Verification summary:

Query not attacker(msg[]) is true.

Query not attacker(password[]) is true.
```

秘匿性 → あり ✓

オフライン攻撃

前方秘匿性

検証方法

オフライン攻撃

盗聴した内容をオフラインで解読すること

- `weaksecret v.`
  秘密値 v のエントロピーが低いとき[3]、
  攻撃者は変数 v に到達可能か

```
(* オフライン攻撃の検証 *)
weaksecret password.
```

[3] 人間が覚えられる程度の文字列しかないとき（パスワードなど）

検証結果

オフライン攻撃

```
$ proverif -color protocol1a.pv
...
The attacker tests whether dec(~M,@weaksecretcst) is fail knowing
~M = enc(msg,password).
This allows the attacker to know whether @weaksecretcst = password.
A trace has been found.
RESULT Weak secret password is false.
...
------------------------------------------------------------
Verification summary:

Weak secret password is false.[4]

Query not attacker(msg[]) is true.

Query not attacker(password[]) is true.
```

秘匿性 → あり ✓
オフライン攻撃 → 可能 ×
前方秘匿性

[4] 弱い秘密を使ったときプロトコルの安全性はない、という意味

検証結果

オフライン攻撃

```
$ proverif -color protocol1a.pv
...
The attacker tests whether dec(~M,@weaksecretcst) is fail knowing
~M = enc(msg,password).
This allows the attacker to know whether @weaksecretcst = password.
A trace has been found.
RESULT Weak secret password is false.
...
------------------------------------------------------------
Verification summary:

Weak secret password is false.[4]

Query not attacker(msg[]) is true.

Query not attacker(password[]) is true.
```

秘匿性 → あり ✓

オフライン攻撃 → 可能 ×

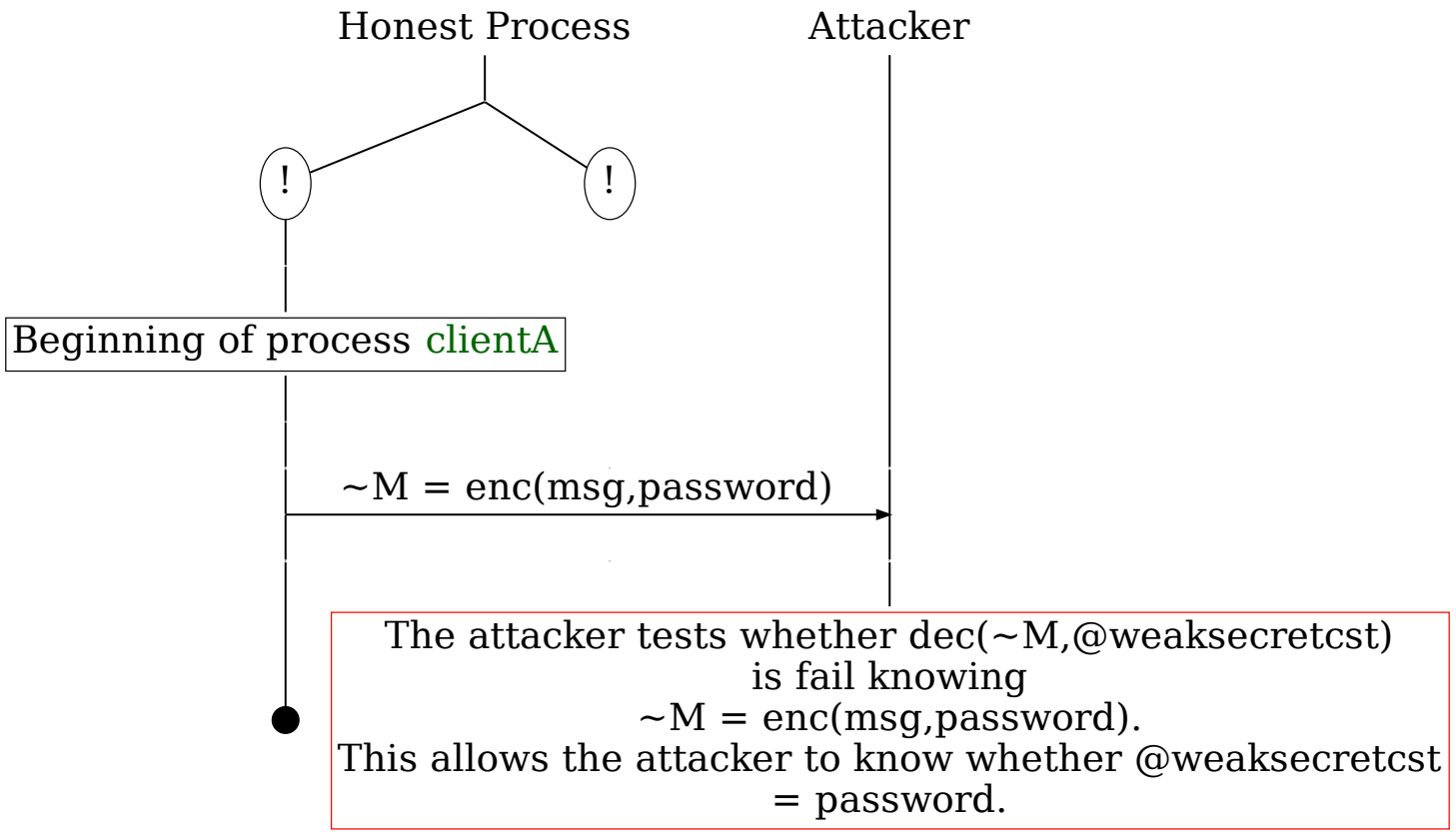
前方秘匿性

[4] 弱い秘密を使ったときプロトコルの安全性はない、という意味

検証結果

オフライン攻撃

A trace has been found.



検証方法

前方秘匿性

秘密鍵が漏洩しても、過去の暗号化通信が復号できないこと

- `phase 1; out(c, password)`

  Phase 0 : パスワードで平文 m を暗号化して送信
  Phase 1 : パスワードを漏洩させる
  パスワード漏洩後に攻撃者は平文 m に到達可能か

```
(* 前方秘匿性の検証 *)
process
  ( (!clientA()) | (!serverB()) | phase 1; out(c, password) )
```

検証結果

前方秘匿性

```
$ proverif -color protocol1b.pv
...
(
    {1}!
    {2}out(c, enc(msg,password))
) | (
    {3}!
    {4}in(c, x: bitstring);
    {5}let recvmsg: bitstring = dec(x,password) in
    0
) | (
    {6}phase 1;
    {7}out(c, password)
)
...
--------------------------------------------------------------
Verification summary:

Query not attacker_p1(msg[]) is false.

Query not attacker_p1(password[]) is false.
```

秘匿性 → あり ✓

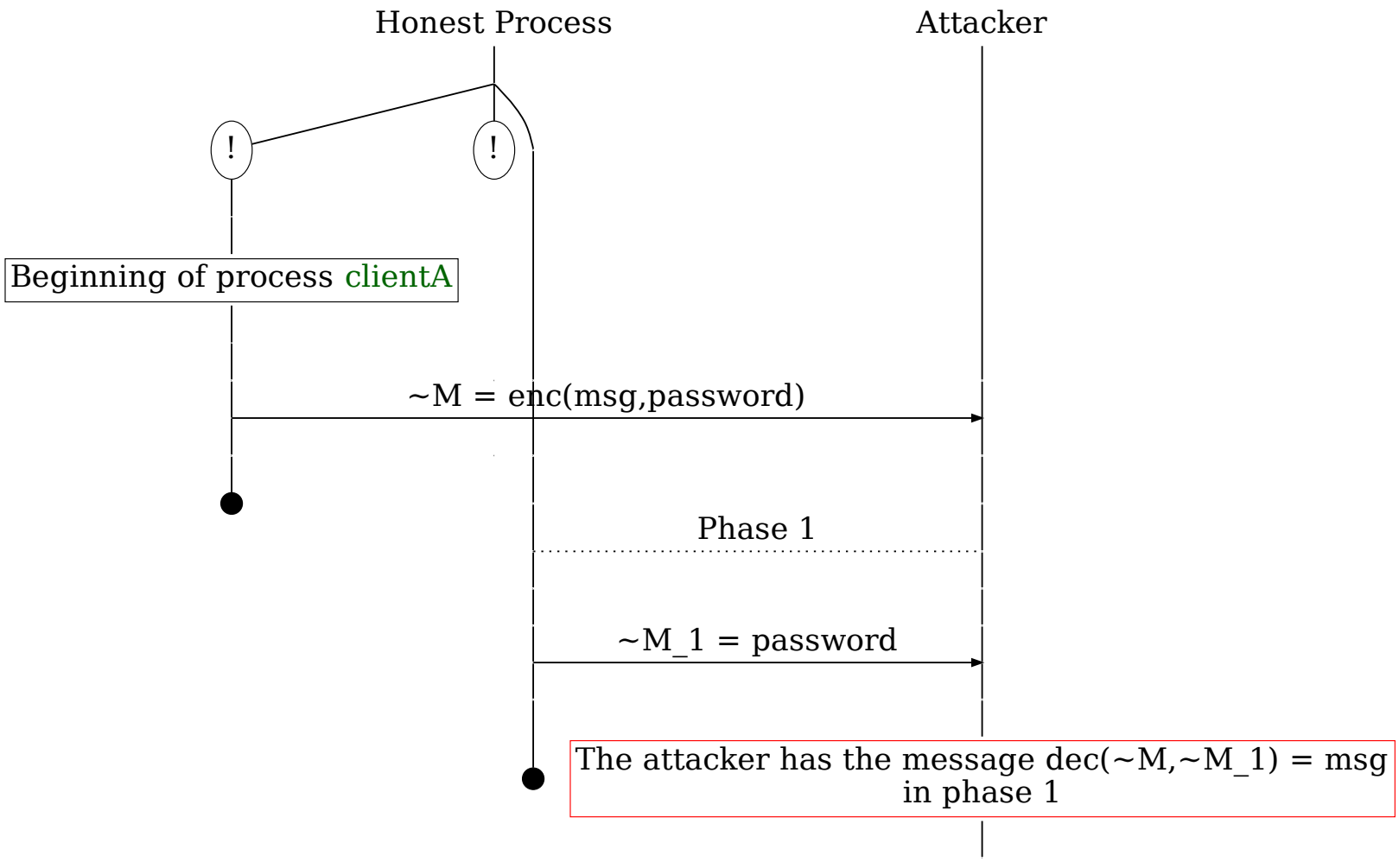
オフライン攻撃 → 可能 ×

前方秘匿性 → なし ×

検証結果

前方秘匿性

A trace has been found.



対策・改善

オフライン攻撃と前方秘匿性

- オフライン攻撃：
  弱い鍵から強い鍵を作る
- 前方秘匿性：
  通信毎に異なる共通鍵を使うようにする

Diffie-Hellman 鍵共有 (DH)

1. Alice は乱数 a を選択する
2. Alice→Bob : $A = g^a \pmod{p}$
3. Bob は乱数 b を選択する
4. Bob→Alice : $B = g^b \pmod{p}$
5. Alice と Bob は共通鍵 K が求まる：
$$K = (g^a)^b = g^{ab} = (g^b)^a \pmod{p}$$

生成元 g と素数 p を適切に選ぶとき、盗聴者は公開値 A, B から共有鍵 K を求めることは困難（離散対数問題）

プロトコルのモデル化

Diffie-Hellman 鍵共有 (DH 鍵共有)

- 一方向：

  $A = g^a \pmod{p}$

  $A = \exp(g, a)$

- 公開値から共通鍵が求まる：

  $K = (g^a)^b = g^{ab} = (g^b)^a \pmod{p}$

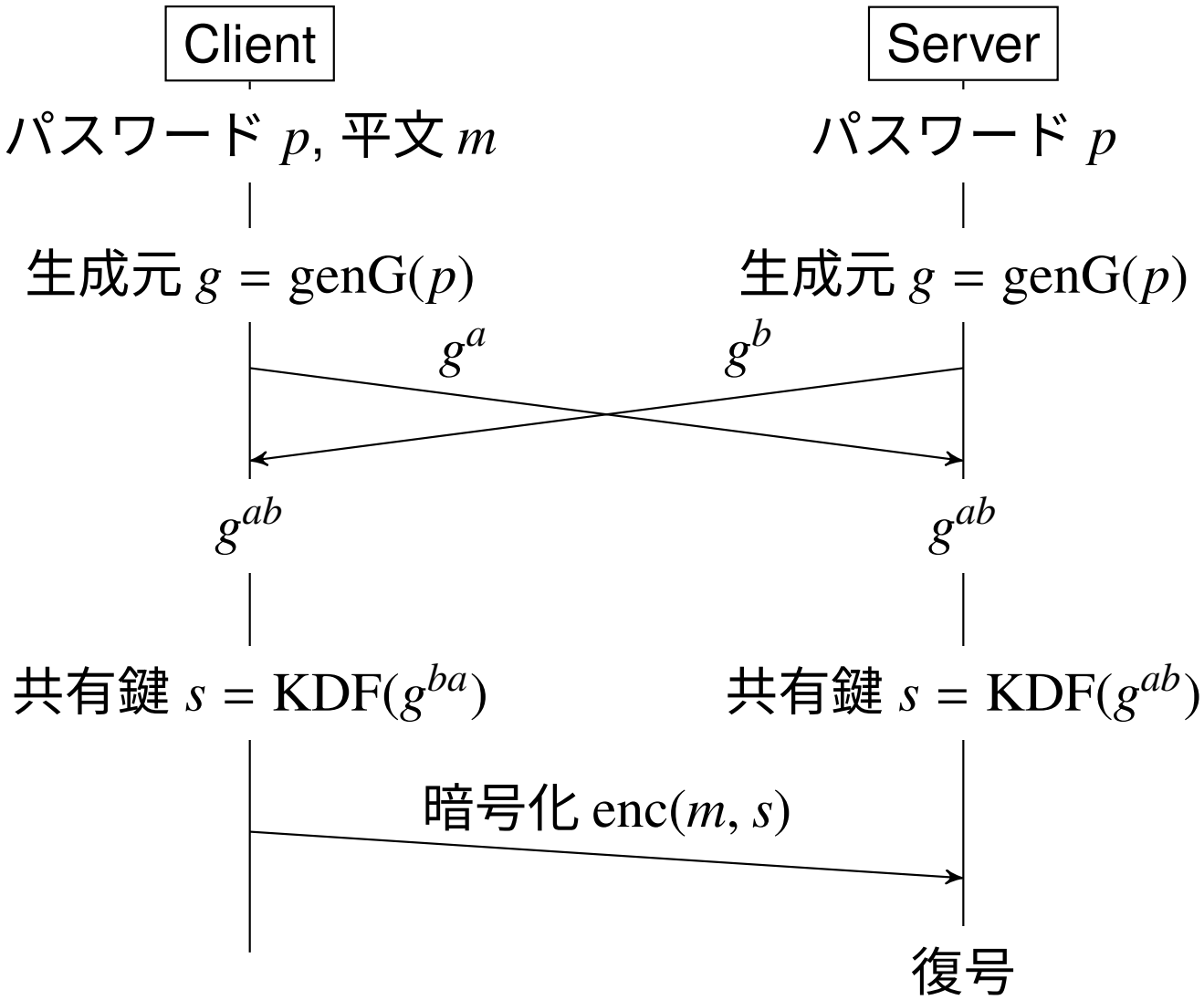
  $K = \exp(\exp(g, a), b) = \exp(\exp(g, b), a)$

ProVerif のコード

```
type G.
type exponent.
const g: G [data].  (* 生成元g *)
fun exp(G, exponent): G.
equation forall a: exponent, b: exponent;
  exp(exp(g,a),b) = exp(exp(g,b),a).
```

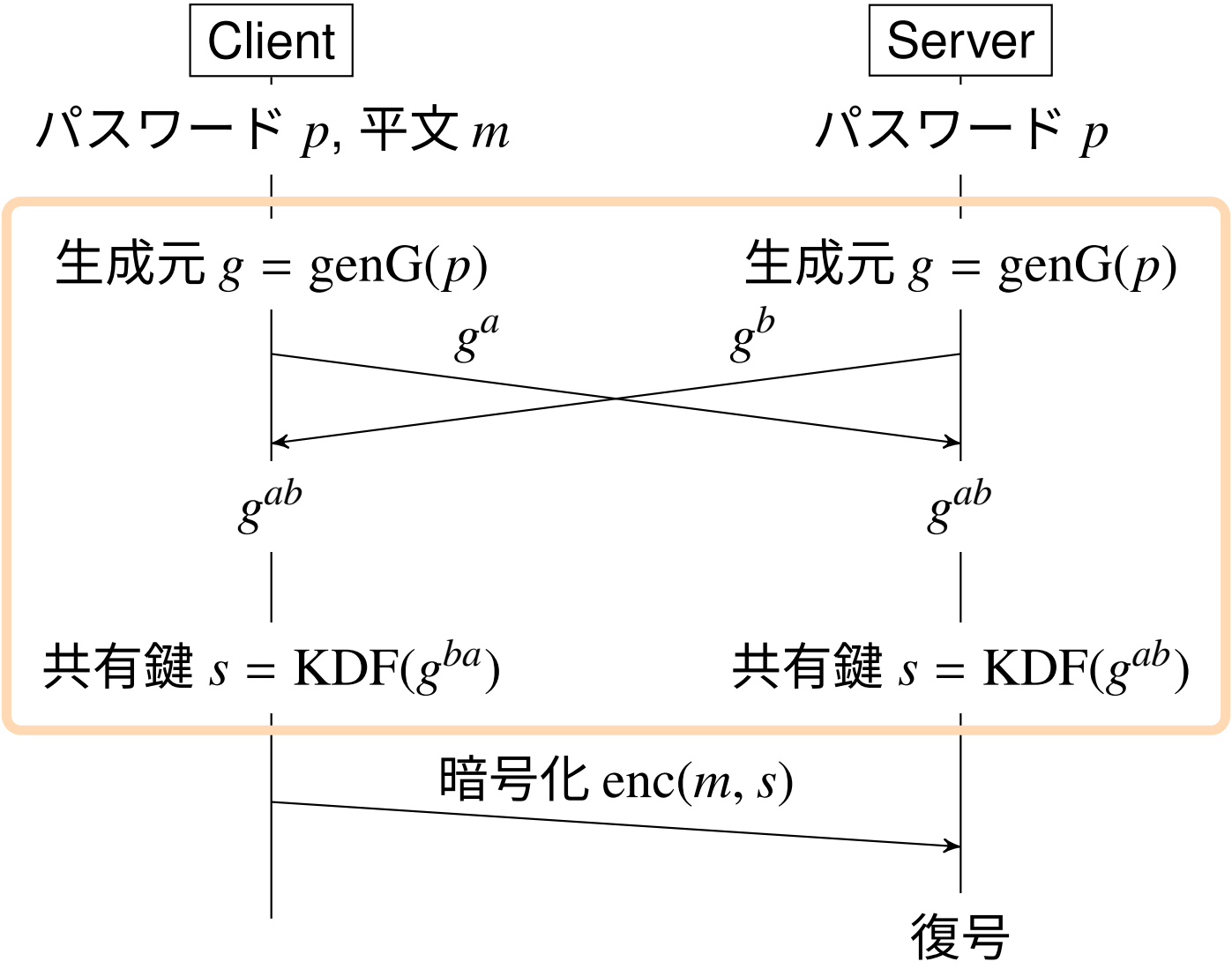
プロトコル β

DH 鍵共有 + 暗号化



プロトコル β

DH 鍵共有 + 暗号化



プロトコルβのモデル化

クライアント–サーバ間の通信

```
let clientA() =
  new randomA: exponent;
  let gA = exp(genG(password), randomA) in
  out(c, gA);
  in(c, gB: G);
  let sharedSecret = KDF(exp(gB, randomA)) in
  let ciphertext = enc(msg, sharedSecret) in
  out(c, ciphertext);
  0.

let serverB() =
  new randomB: exponent;
  let gB = exp(genG(password), randomB) in
  in(c, gA: G);
  out(c, gB);
  let sharedSecret = KDF(exp(gA, randomB)) in
  in(c, ciphertext: bitstring);
  let recvmsg = dec(ciphertext, sharedSecret) in
  0.

process
  ( (!clientA()) | (!serverB()) )
```

検証方法

秘匿性、オフライン攻撃、前方秘匿性

- `attacker(v)`
  攻撃者は変数 v に到達可能か
- `weaksecret v.`
  秘密の値 v のエントロピーが低いとき、
  攻撃者は変数 v に到達可能か
- `phase 1; out(c, password)`

  Phase 0 : パスワードで平文 m を暗号化して送信
  Phase 1 : パスワードを漏洩させる
  パスワード漏洩後に攻撃者は平文 m に到達可能か

検証結果

秘匿性、オフライン攻撃

```
$ proverif -color protocol2a.pv
...
--------------------------------------------------------------
Verification summary:

Query not attacker(msg[]) is true.

Query not attacker(password[]) is true.

Weak secret password is true.
```

秘匿性 → あり ✓
オフライン攻撃 → 困難 ✓
前方秘匿性

検証結果

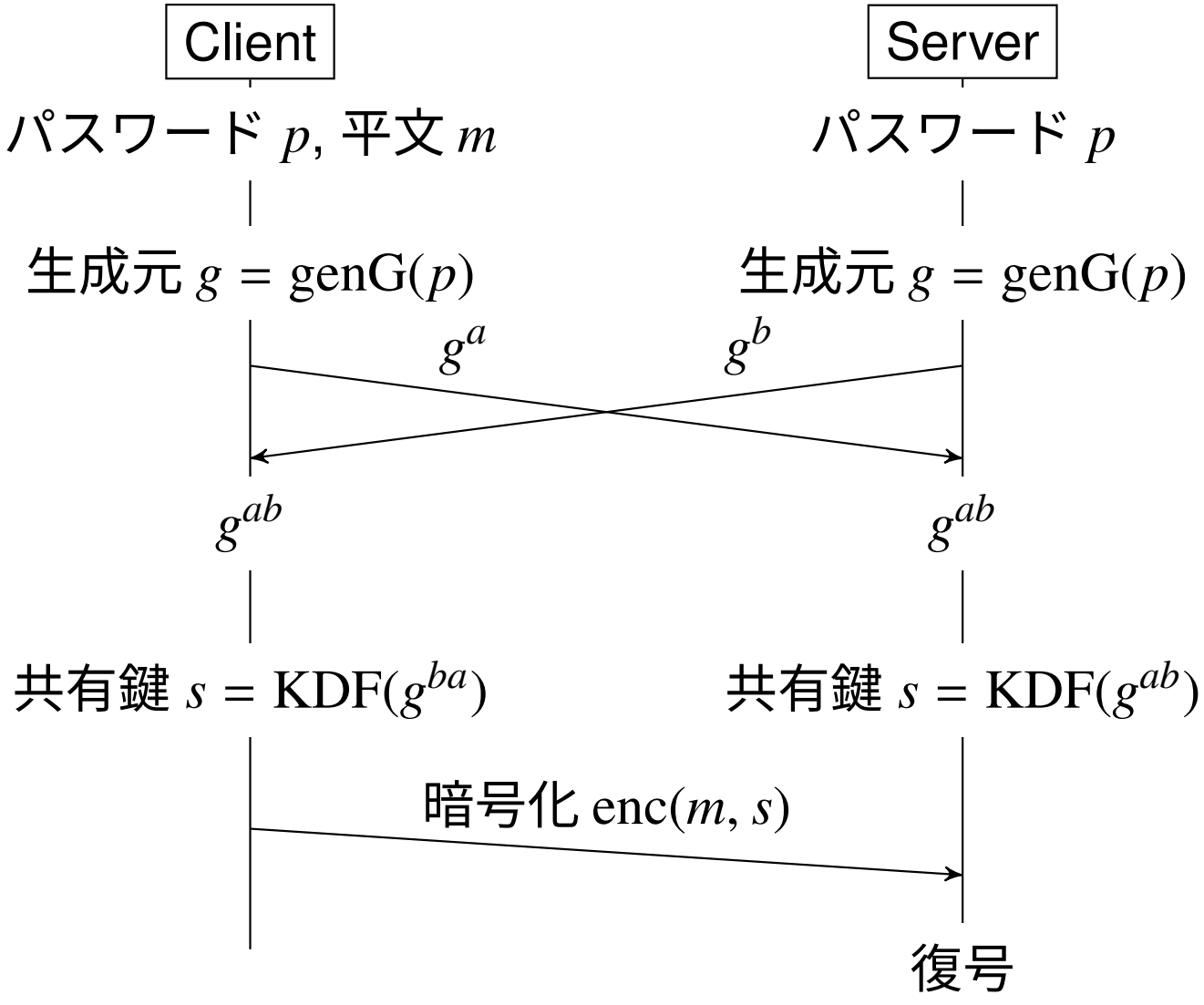
前方秘匿性

```
$ proverif -color protocol2b.pv
...
--------------------------------------------------------------
Verification summary:

Query not attacker_p1(msg[]) is true.

Query not attacker_p1(password[]) is false.

...
```

秘匿性 → あり ✓
オフライン攻撃 → 困難 ✓
前方秘匿性 → あり ✓

プロトコル β

Wi-Fi の新規格 WPA3 を参考に作成



おわりに

- ProVerif は秘匿性や真正性を自動で検証可能
- いろんなプロトコルを検証してみると楽しい

Happy ProVerifying!

参考文献 I

- Blanchet at el.: ProVerif 2.02pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. INRIA, September 2020.
- Blanchet: ProVerif Automatic Cryptographic Protocol Verifier User Manual for Untyped Inputs. INRIA, September 2020.