n チャネルメッセージ伝送方式による暗号化通信

木下研究室　200602825　栗山 知也

従来の暗号方式～公開鍵暗号～

第三者機関による
公開鍵の正当性証明が必要

nチャネルメッセージ伝送

n本の通信路を使用する伝送方式

事前の鍵が不要！！
第三者機関も必要ない！！

研究の概要

・従来の暗号方式（公開鍵暗号）と　n チャネルメッセージ伝送

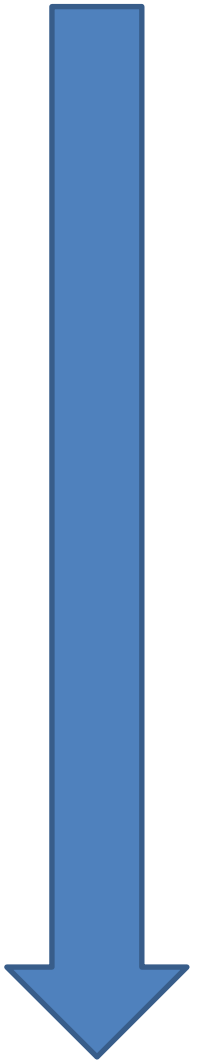
・PSMT：Perfectly Secure Message Transmission

・ASMT：Almost Secure Message Transmission

・Basic プロトコル

信量の改善

・改良プロトコル



安全性の定義
PSMT：Perfectly Secure Message Transmission

盗聴や改ざんする敵が **n** 本の通信路の内 **t** 本に潜んでいるとき・・・・

1. **盗聴耐性**
   敵は送信メッセージに関する情報を何も得られない。

2. **改ざん耐性**
   受信者がメッセージを正しく受信できる確率が 100 % で
   ある。

安全性の定義
ASMT：Almost Secure Message Transmission

1. **盗聴耐性**
   敵は送信メッセージに関する情報を何も得られない。

2. **改ざん耐性**
   受信者がメッセージを正しく受信できる確率が 1-δ 以上である。

3. **失敗検知能力**
   受信者が正しく受信できない確率が δ 以下であり、そのとき受信者は failure を出力できる。

4. **遮断耐性**
   敵が t 本の通信路を遮断しても受信者は残りの通信路で得た情報だけからメッセージを受信できる。

PSMT と ASMT の比較

PSMT		ASMT
2-round	1-round	1-round
$n \geqq 2t + 1$	$n \geqq 3t + 1$	$n \geqq 2t + 1$

n : 通信路の数
t : 敵が盗聴改竄できる通信路の数

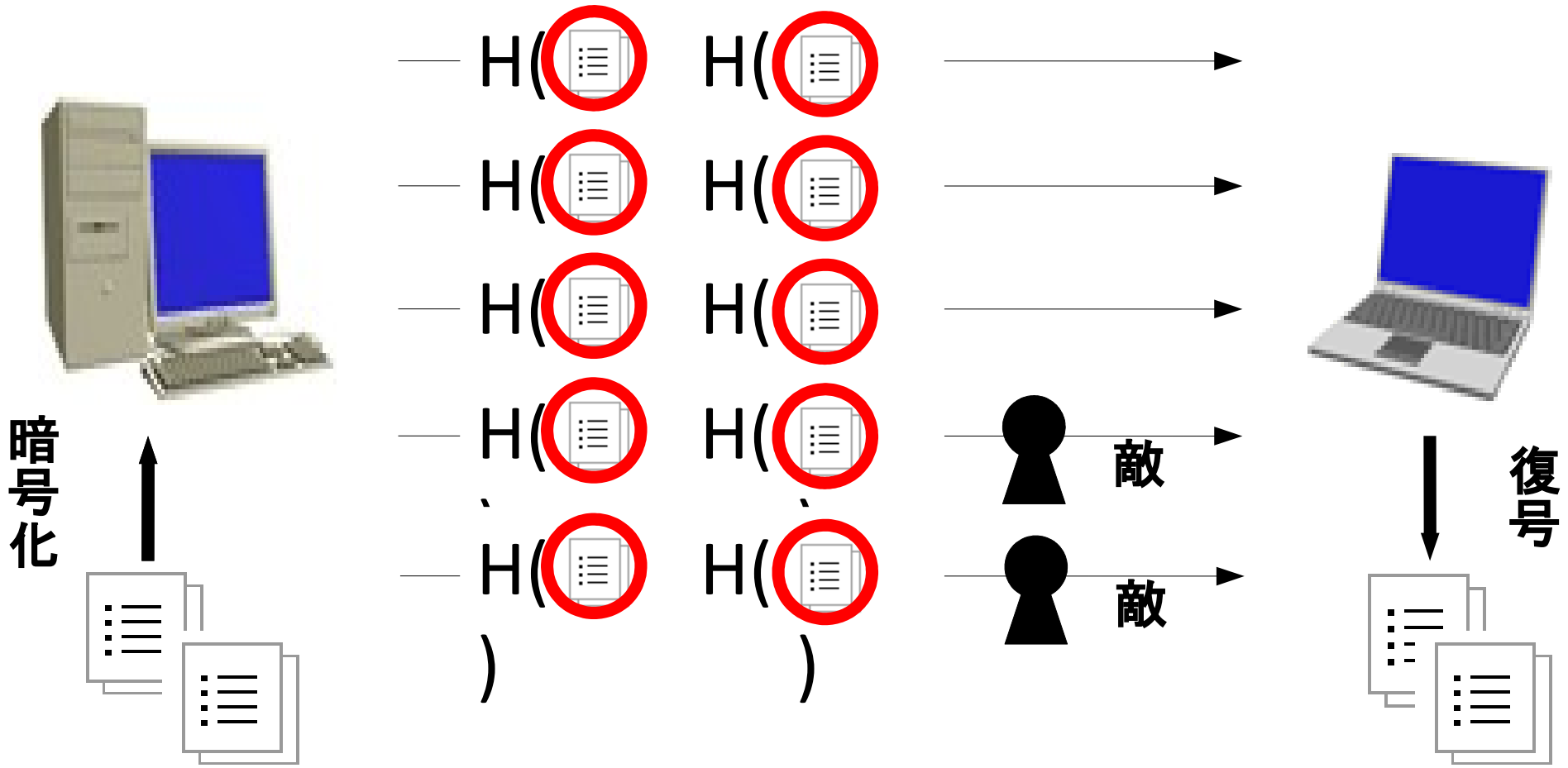
ASMT の歴史

2004　Srinathan et al.
プロトコル提案　→　間違いがあった。

2007　Kurosawa et al.
ASMT を厳密に定義。n=2t+1 のとき通信効率の限界を示し、限界に近いプロトコル提案。ただし計算量は指数関数的。

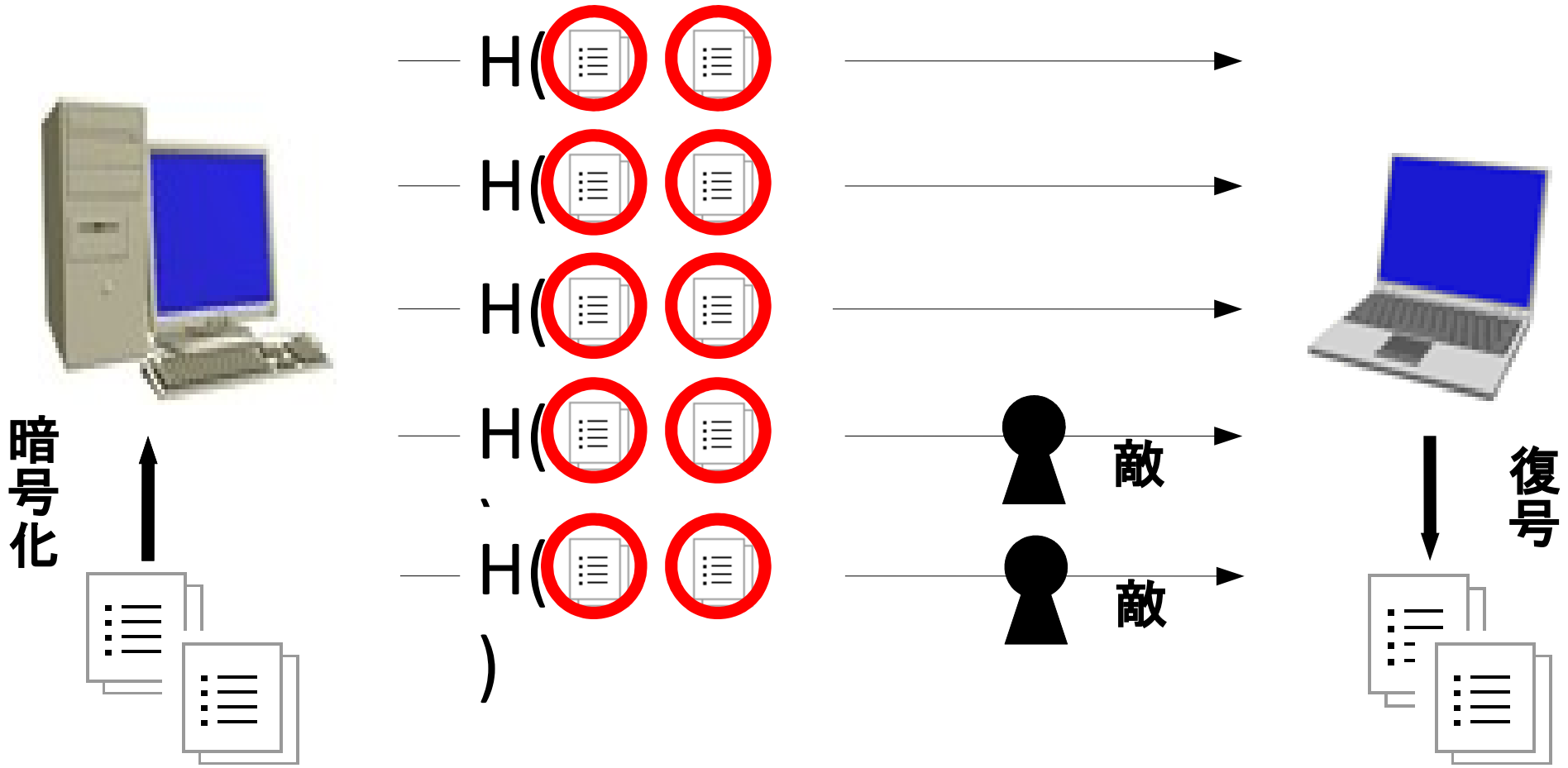
2008,2009　木下研究室
Basic プロトコルを提案実装。計算量は多項式時間。
更に計算量はそのままで通信量を改良したプロトコルを提案実装。

Basic プロトコルのアイデア



送信したい秘密情報を暗号化する
ハッシュ関数に入れて送信する

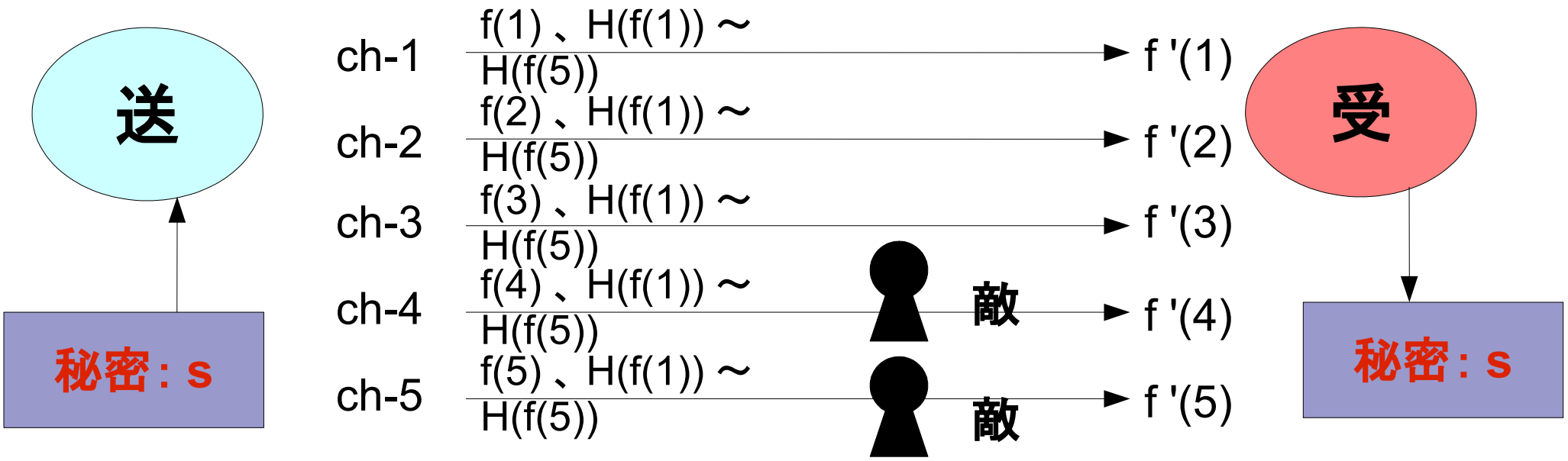
改良プロトコルのアイデア



送信したい秘密情報を暗号化する
ハッシュ関数にまとめて入れて送信する

ハッシュ値の長さは固定なので通信量削減

Basic プロトコル



n=2t+1（n：通信路の数　t：敵の数）

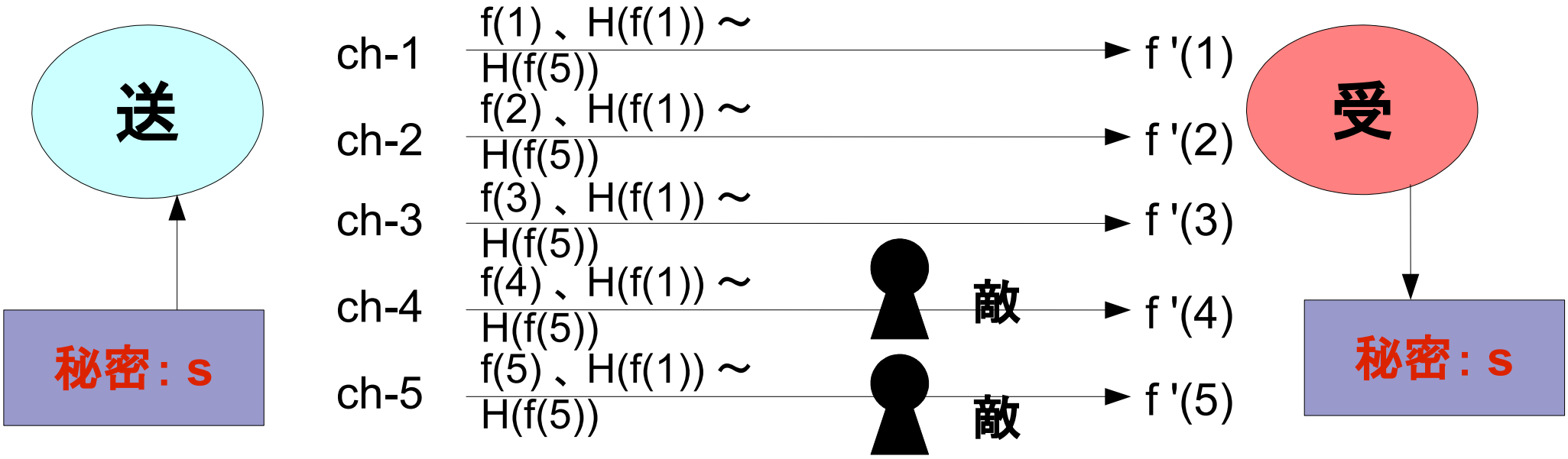
送信者

・$f(x)=s+a_1x+a_2x^2+\cdots+a_tx^t \bmod P$ をランダムに作る。

・ハッシュ値 H(f(1)) ～ H(f(n)) を計算する。

・各 ch-i に f(i)、H(f(1)) ～ H(f(n)) を送る。

Basic プロトコル



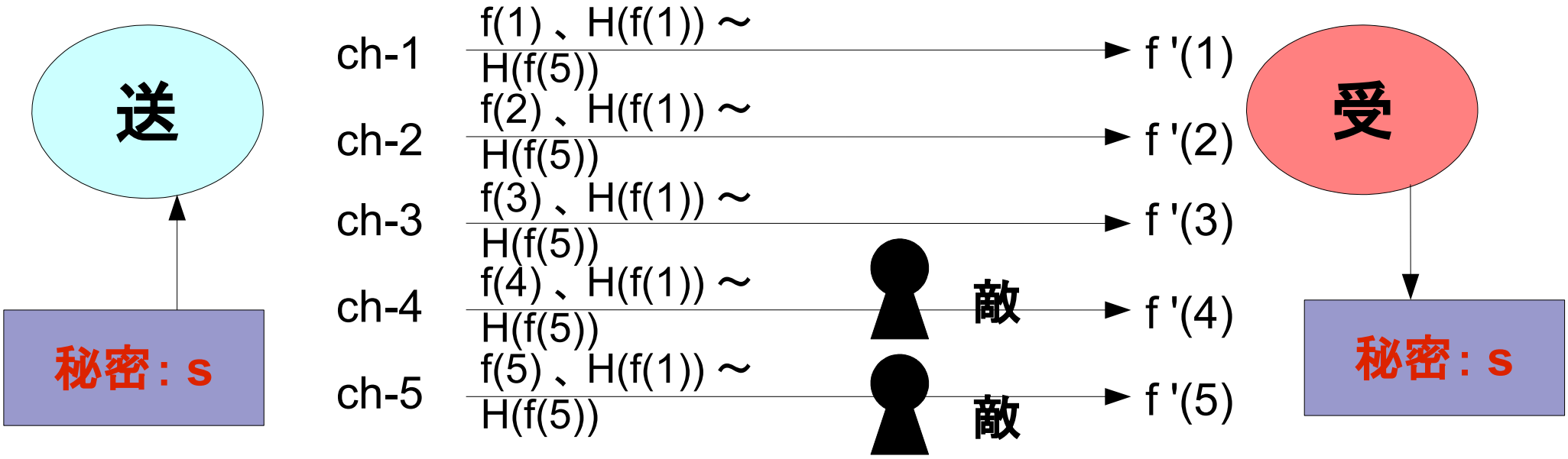
n=2t+1（n：通信路の数　t：敵の数）

敵

・f(1) ～ f(n) のうち、t 個しか知らない。
→ f(x) は t 次関数なので t 点からは s について何も分からない。

・ここでハッシュ関数 H は一方向性があると仮定するので H(f(x)) から f(x) を
逆算できない。
→ハッシュ値からは s について何も分からない。

Basic プロトコル



n=2t+1（n：通信路の数　t：敵の数）

受信者

- f '(1) ～ f '(n) を得る。
- 多数決で正しい H(f(1)) ～ H(f(n)) を得る。
- H(f '(1)) ～ H(f '(n)) を計算し、H(f(1)) ～ H(f(n)) と等しいか調べる。
- H(f(i))=H(f '(i)) となる f(i) は t+1 個以上ある。
  → f(x) を復元し、s を得られる。

Basic プロトコルの通信効率

n =2t+1 の通信効率の限界は Xi を ch を流れる情報の集合、δ を失敗確率とすると

$$|X_i| \geq (|S| - 1)/\delta + 1$$

であった。これより

秘密 S の長さが q ビットなので f(i) も q ビットであり、ハッシュ値は h ビットであるので通信効率の限界は q+hn ビットとなる。

$$q + hn = \log_2 |X_i| \geq h + q - \log_2 t$$

両辺を比較すると左辺の hn と右辺の h の差が大きいことがわかる。従って、Basic プロトコルは通信効率の限界に近いとは言えない。

改良プロトコルの通信効率

一度に送る Si の個数を m とすると、改良プロトコルの通信効率は

$$qm+ hn= \log_2 |X_i| \geq h+ qm- \log_2 t$$

q=h,m=n とすると

$$2hn= \log_2 |X_i| \geq h+ hn- \log_2 t > h(n+ 1)- \log_2 n$$

オーダで評価すると

$$O(hn)= O(\log_2 |X_i) \geq \Omega\ (h(n+ 1)- \log_2 n)$$

ここで n を十分に大きい値だとすると

$$O(hn)= O(\log_2 |X_i) \geq \Omega\ (h(n+ 1)- \log_2 n) \rightarrow\ \Omega\ (hn)$$

となり、通信効率の限界に近いと言える。

改良プロトコルの実装

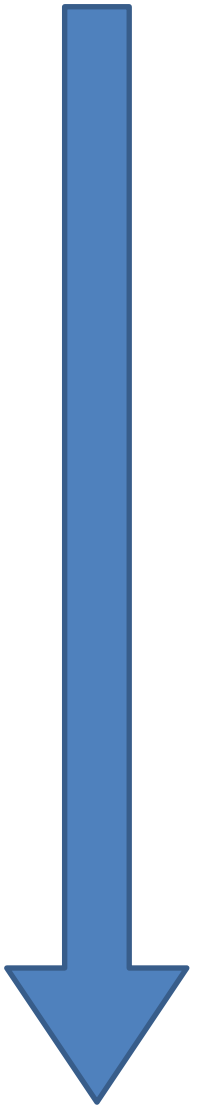
暗号のライブラリが充実している Flash(ActionScript) を選択

ハッシュ関数に SHA-1 を選択

ネットワーク上で実装するために多倍長に対応

暗号、復号をオブジェクトにして処理を分割した

ネットワーク上の実装の前段階として仮想ネットワーク上での実装を次の目標とした



プログラムの実行例


秘密情報　半角16進数で750bit未満の数を入力してください
e30812fdc51564560f5132f7821e19811021af01b8557fafec1db4754b5e6fc30c0b703d8763e494385bd959306af27ba3df2e59983ac2
計算
クリア
f_receive[1][2]=52d34ba38e406431ab0a8c774b1bd9af8a034d7c7e5bad7ac66a956ce72bca76
f_receive[1][3]=2f8528807e50a6d830763ea0113bf0af2d0a6a7bbb0519be9b629ee9d149a14e
f_receive[1][4]=50afb7a6cfea57f29c3aa6ca3e732d7ec12298b1400822bd1ef0018d10bc6fa4
f_receive[2][0]=5022277991e87a092046759e04e5468be328ce639e61b27500251a9f51d30aed
f_receive[2][1]=1e1808a78ea88291239414b3014eb41dd04dd14da662cf4c54c2d11a991ce5d1
f_receive[2][2]=198cd61391925ca4572444a46b16312c787cb9b352eb41e213698aa51622c9d8
f_receive[2][3]=423c1261d7fef49ab43f60a89a3eb37599eca4bb84a9bb0c9f8a9f085e18eb9c
f_receive[2][4]=19d5598c0bb2d53722ba659e9e56bf909af4bb9dd5dbdec01a9a1d62a6b262df
送信者が作成したH_calculate[i]
H_calculate[0]=2208cce57c107355b4e30d241d26dd136ee82efd
H_calculate[1]=2dcda50bbdda2205374efb70d62623c431ee6793
H_calculate[2]=75d34c8bdc69fc815f5e61f42db085a79684a96a
H_calculate[3]=af7770b641ada0fe2b72a629c6f973d11633b915
H_calculate[4]=947266bfa5c7b330ff5e0af77d65169b6f3de2df
ハッシュ値の等しいチャンネルは
1,3,5
正常に終了しました
復号結果
e30812fdc51564560f5132f7821e19811021af01b8557fafec1db4754b5e6fc30c0b703d8763e494385bd959306af27ba3df2e59983ac2
受信したハッシュ値
計算したハッシュ値

結論

・提案プロトコルにより以前に比べ通信効率の限界に近い通信量に改善できた。

・改良プロトコルを Flash(ActionScript) で実装した結果、正しく動作することが検証できた。

・ネットワークでの実装に近づけた。また、仮想ネットワークでの実装方法を提案した。