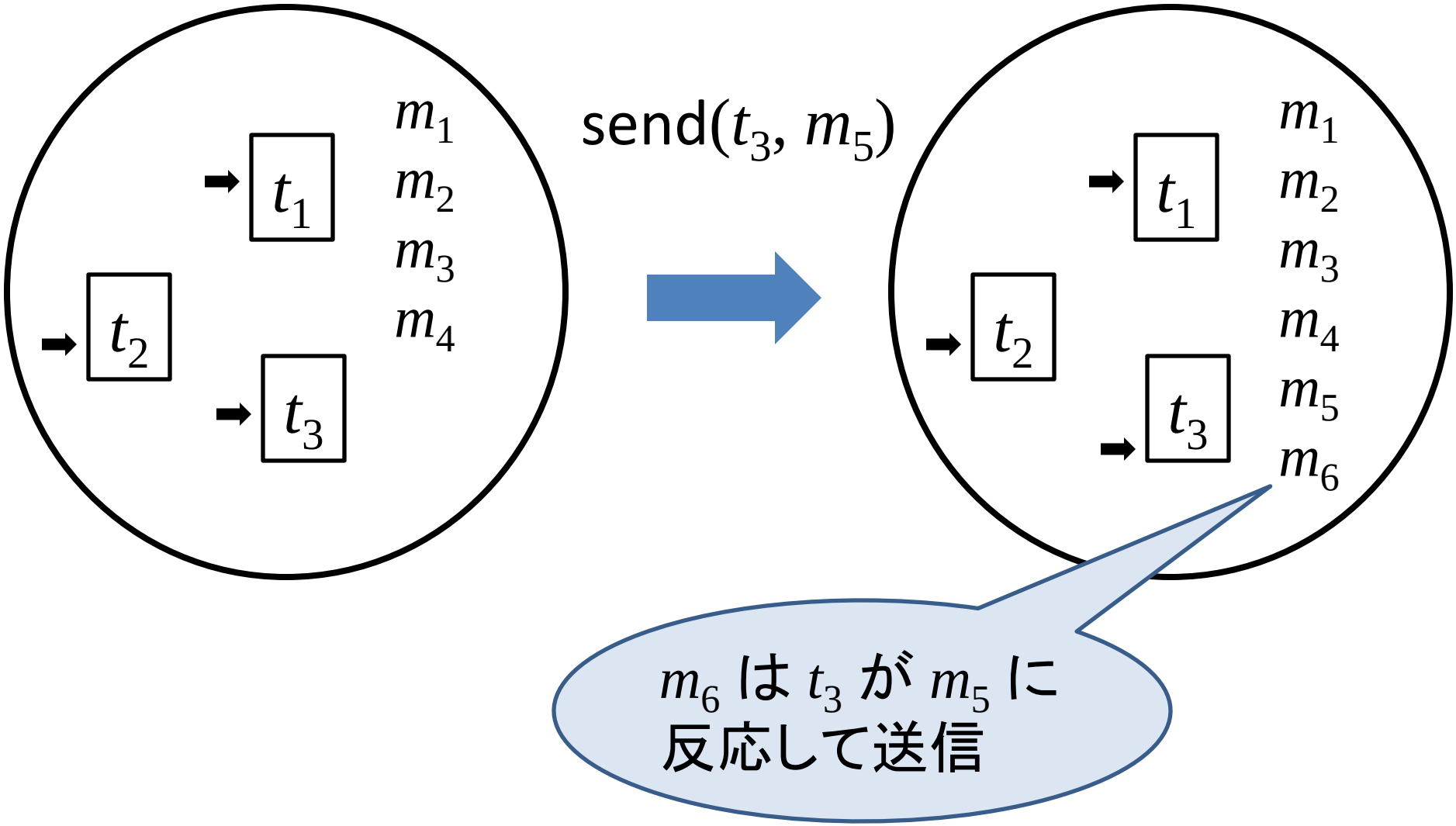
Mapping Lemma

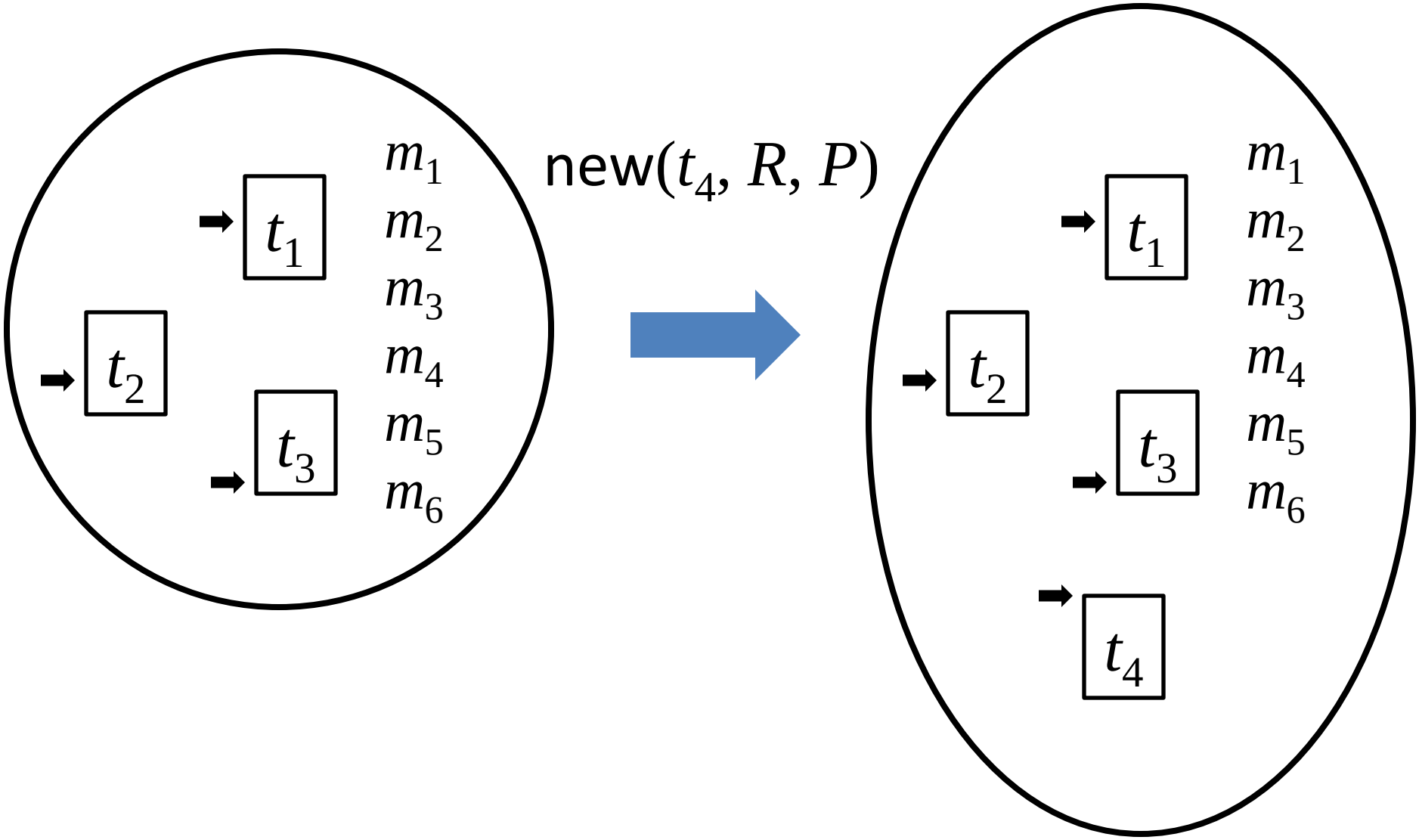
動機

- Abadi-Rogaway --- 受動的な攻撃者のみ
- 能動的な攻撃者は？
- たとえば、NSL プロトコルの相互認証性は、計算論的にも成り立つか？
  - 記号論的には成り立つ。
    - たとえば、ストランド空間による議論
- 計算論的な実行過程が記号論的な実行過程に対応するならば、記号論的な正当性から計算論的な正当性が導かれる。

プロトコルの実行モデル

- 攻撃者は次のような命令を順に実行する。
  - send(t, m): スレッド t にメッセージ m を送る。
  - new(t, R, P): ロール R の正規参加者をパラメータ P で起動してスレッド t とする。
- 正規参加者のスレッドの内部状態
  - プロトコル（自分のロール）の中の実行位置
  - パラメータや変数の値
- システム全体の状態
  - 正規参加者のスレッドの内部状態の集合
  - ネットワークに送信されたメッセージの集合
  - 攻撃者の内部状態


m_1
m_2
m_3
m_4
t_1
t_2
t_3
send(t_3, m_5)
m_1
m_2
m_3
m_4
m_5
m_6
t_1
t_2
t_3
m_6 は t_3 が m_5 に
反応して送信


t_1
t_2
t_3
m_1
m_2
m_3
m_4
m_5
m_6
new(t_4, R, P)
t_1
t_2
t_3
t_4
m_1
m_2
m_3
m_4
m_5
m_6

計算論的モデルと記号論的モデル

	計算論的	記号論的
スレッド	番号（ビット列）	名前
メッセージ	ビット列	項
定数	タグ付きビット列 $\langle \mathrm{bit}, 0 \rangle, \langle \mathrm{bit}, 1 \rangle$	定数 $0, 1$
参加者名	タグ付きビット列 $\langle \mathrm{id}, a \rangle$	名前（定数） A
鍵・ノンス	タグ付きビット列 $\langle \mathrm{key}, k \rangle, \langle \mathrm{nonce}, n \rangle$	名前（定数） K, N
対	タグ付きビット列 $\langle \mathrm{pair}, m_1, m_2 \rangle$	構成子による項 $\langle m_1, m_2 \rangle$
暗号文	タグ付きビット列 $\langle \mathrm{enc}, \langle \mathrm{key}, k \rangle, m \rangle$	構成子による項 $\{m\}_K^r$
パラメータ・変数の値	ビット列	項

暗号化スキーム

- ここでは公開鍵の暗号化スキームを用いる。
- IND-CCA2 を満たすと仮定。

ランダムネス記号

- 暗号化は乱数化されているので、平文は同じでも異なる暗号文を区別するために、記号論的モデルでは、ランダムネス記号 r を導入する。

$$\{m\}_K^r$$

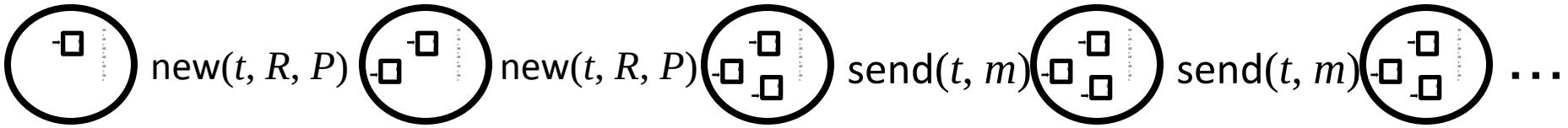
- 個々のランダムネス記号は、唯一の暗号文に付随する。

正規参加者が可能な操作

- 暗号化
- 復号（秘密鍵を有している場合）
- 同じかどうかの検査 ⇒ 条件分岐
  - 記号論的：ランダムネス記号も含めて同じか？
  - 計算論的：ビット列として同じか？
- ノンスの生成
- 定数
- 対（および射影）
- 項の種類の検査 ⇒ 条件分岐

(計算論的および記号論的)トレース

- システムの状態と攻撃者の命令が交互に並んだもの



- 正規参加者はプロトコルの記述に従って実行される。
- 攻撃者
  - 計算論的には、確率的多項式時間チューリング機械によって実現される。
  - 記号論的には、Dolev-Yao モデルに従う。
    - ストランド空間の攻撃者

Mapping Lemma

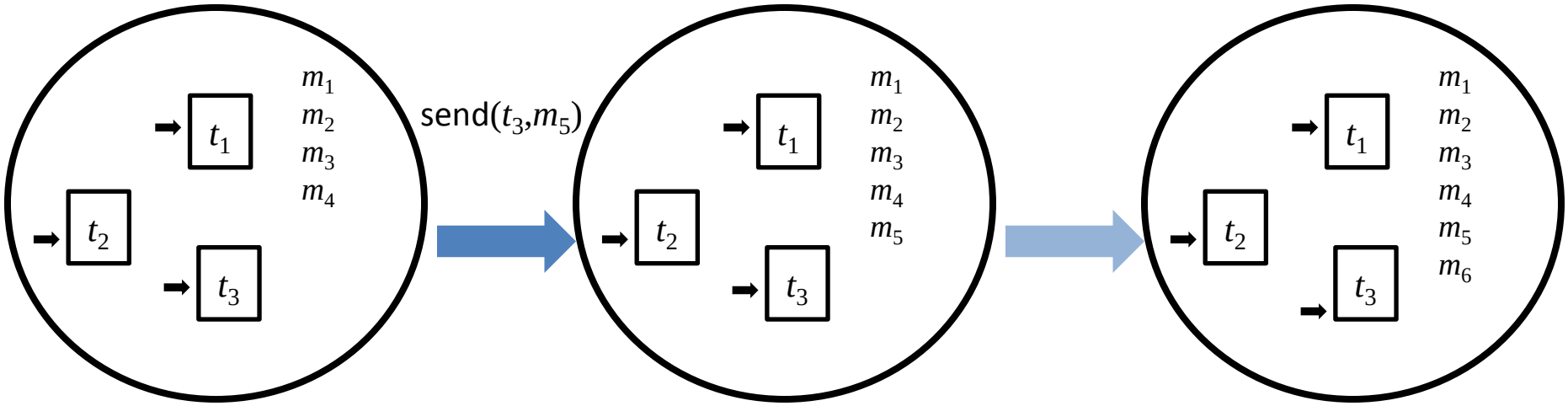
- 計算論的なトレースに対して、記号論的なトレースが対応する。
  - ただし、無視できるくらい小さい確率で、計算論的なトレースは、記号論的なトレースから逸脱することがあり得る。
- 実際には、計算論的な攻撃者に対して、記号論的にプロトコルを実行しながら、記号論的なトレースを構成する。

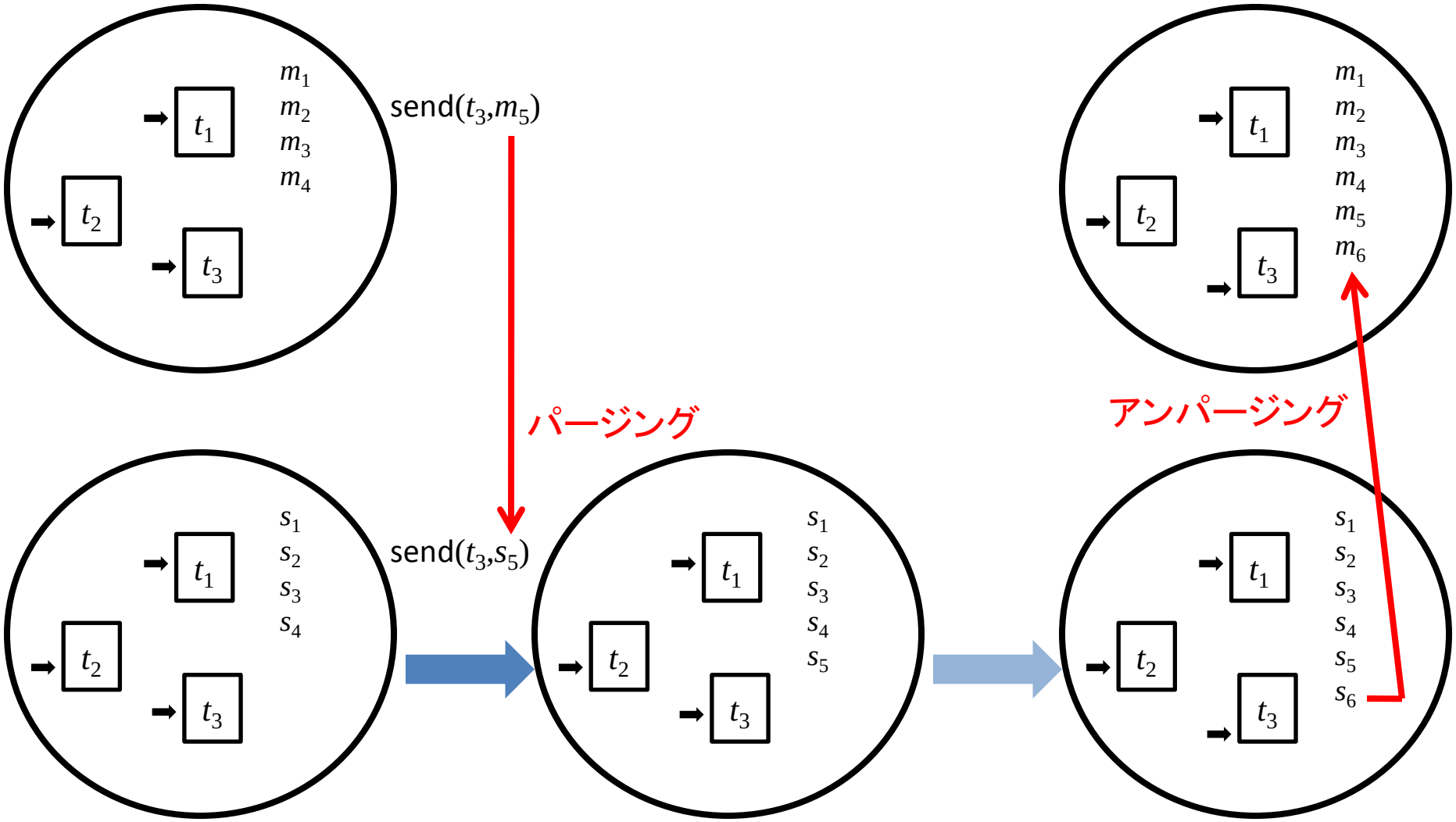
初期化

- 記号論的モデルにおける鍵や参加者名には、あらかじめ、対応するビット列が割り当てられているとする。これを τ で表す。
  - 鍵 K に対して、$\tau(K) = \langle \mathrm{key}, k \rangle$
  - 参加者名 A に対して、$\tau(A) = \langle \mathrm{id}, a \rangle$
- ノンスは動的に生成されるので、ビット列との対応は別に扱う。
- 各参加者と攻撃者の知識（持っている鍵など）も定める。

計算論的な実行のシミュレーション

- 正規参加者は記号論的に実行する。
  - 記号論的な初期状態を用意する。
  - その中の記号に対しては τ を定義する。
- 攻撃者は計算論的（チューリング機械）
  - 攻撃者の send(t, m) 命令に対しては、ビット列 m をパージングする。その結果に従って、スレッド t を記号的に実行する。t が送信するメッセージは、アンパーシングによりビット列に変換して攻撃者に送る。
  - 攻撃者の new 命令に対しては、指定されたロールの記号論的なスレッドを生成。


t_1
t_2
t_3
m_1
m_2
m_3
m_4
send(t_3,m_5)
t_1
t_2
t_3
m_1
m_2
m_3
m_4
m_5
t_1
t_2
t_3
m_1
m_2
m_3
m_4
m_5
m_6


m_1
m_2
m_3
m_4
t_1
t_2
t_3
send(t_3,m_5)
パージング
m_1
m_2
m_3
m_4
m_5
m_6
アンパージング
s_1
s_2
s_3
s_4
send(t_3,s_5)
s_1
s_2
s_3
s_4
s_5
s_1
s_2
s_3
s_4
s_5
s_6

パージング

- ビット列はタグ付されている。
  - $\langle \mathrm{id}, a \rangle$
  - $\langle \mathrm{key}, k \rangle$
  - $\langle \mathrm{nonce}, n \rangle$
  - $\langle \mathrm{bit}, b \rangle$
  - $\langle \mathrm{pair}, m_1, m_2 \rangle$
  - $\langle \mathrm{enc}, \langle \mathrm{key}, k \rangle, m \rangle$
- ビット列と項の対応を記録する表を保持する。
  - パージングとアンパージングの環境

パージング

- parse(m) = s

  if (m, s) は記録されている
- parse(⟨id, a⟩) = τ^{-1}(⟨id, a⟩)
- parse(⟨key, k⟩) = τ^{-1}(⟨key, k⟩)
- parse(⟨nonce, n⟩) = N

  ノンス記号 N を新たに生成

  (⟨nonce, n⟩, N) を記録
- parse(⟨bit, b⟩) = b
- parse(⟨pair, m_1, m_2⟩) = ⟨parse(m_1), parse(m_2)⟩

パージング

- parse(⟨enc, ⟨key, k⟩, m⟩) = $\{ s \}_K^r$

  s = parse(dec(k^{-1}, m)) とする

  - k^{-1} は k に対応する秘密鍵
  - dec は復号アルゴリズム

  τ^{-1}(⟨key, k⟩) = K とする

  ランダムネス記号 r を新たに生成

  (⟨enc, ⟨key, k⟩, m⟩, $\{ s \}_K^r$) を記録

細かい話

- $\mathrm{dec}(k^{-1}, m)$ において、復号に失敗したり、復号した結果がビット列の形式に合わない（たとえばタグが正しくない）場合、

$$\mathrm{parse}(\langle \mathrm{enc}, \langle \mathrm{key}, k \rangle, m \rangle) = \{ \mathrm{ERROR} \}_K^r$$

とする。

- 記号論的実行においては、このような項を復号しようとすると、エラーになるものとする。
- 記号論的にも計算論的にも、暗号文を暗号化に使った公開鍵に対応しない秘密鍵で復号しようとするとエラー。

アンパージング

- unparse(s) = m

  if (m, s) が記録されている
- unparse(A) = $\tau(A)$
- unparse(K) = $\tau(K)$
- unparse(N) = ⟨nonce, n⟩

  N は新たに作られたノンス記号のはず

  ビット列 n をランダムに生成

  (⟨nonce, n⟩, N) を記録
- unparse(b) = ⟨bit, b⟩
- unparse(⟨s_1, s_2⟩) = ⟨pair, unparse(s_1), unparse(s_2)⟩

アンパージング

- unparse($\{ s \}_K^r$) = m = enc(k, unparse(s))

  $\tau(K) = \langle \mathsf{key}, k \rangle$ とする

  r は新たに作られたランダムネス記号のはず

  $(m, \{ s \}_K^r)$ を記録

シミュレーションの正当性

- 計算論的な攻撃者に対してプロトコルを(計算論的に)直接に実行するのと、パージングとアンパージングによるシミュレーションは、無視できる確率を除いて一致する。
- 不一致が起こるのは、ノンスの生成もしくは暗号化の際に、それまでに現れたビット列と衝突する場合。
  - 衝突が起こると、ノンスもしくはランダムネス記号の新規性が成り立たなくなる。(同じビット列に複数の記号が割り当てられてしまう。)
  - この確率は無視できるほど小さい。

なぜなら

- ビット列はタグ付けされているので、ノンスと暗号文に分けて考えてよい。
  - ノンスはランダム性よりぶつかることはない。
  - 暗号文は用いた鍵が付加されているので、鍵ごとに考えればよい。
    - 異なる平文の暗号文がぶつかることはない。
    - 同じ平文 m の暗号文がぶつかることがあるならば、IND-CCA2 の仮定に反する。m_0=m と m_1 を LR オラクルに送り、さらに m を暗号化して LR オラクルの結果と一致すれば、b=0 と推測できる。

Mapping Lemma

- 暗号化スキームが IND-CCA2 安全ならば、シミュレーションによって得られる記号論的なトレースの攻撃者は Dolev-Yao モデルに従う。
- すなわち、攻撃者が送信したビット列をパージングして得られる項 s は、それまでにネットワークに送信された記号論的なメッセージと攻撃者の知識から導出できる。
- Micciancio-Warinschi（2004）

証明スケッチ

- 定理の結論が成り立たないと仮定すると、項 s は攻撃者の知りえないノンスもしくは暗号文 m を、もともとネットワークに送信されたのとは異なる形で含む。
  - 何段かに暗号化されている場合もある。
  - m はもともとは暗号化されていたはず。

たとえば

- $\{A, N_A\}_{KB}{}^{r_1}$ が送信されていたときに、以下のようなメッセージは Dolev-Yao に反する。

  $N_A \quad \{B, N_A, N_B\}_{KA}{}^{r_2} \quad \{A, N_A\}_{KB}{}^{r_3}$

  $\{\{N_A\}_{KA}{}^{r_4}\}_{KB}{}^{r_5}$

- これに対して、以下のようなメッセージは OK。

  $\{A, N_A\}_{KB}{}^{r_1} \quad \{\{A, N_A\}_{KB}{}^{r_1}\}_{KA}{}^{r_6}$

証明スケッチ

- 正規参加者は高々多項式 $p(\eta)$ 個のノンスもしくは暗号文を生成するものとする。(η はセキュリティ・パラメタ)
- プロトコルの攻撃者 A0 から、拡張された IND-CCA2 ゲームの攻撃者 A を構成する。

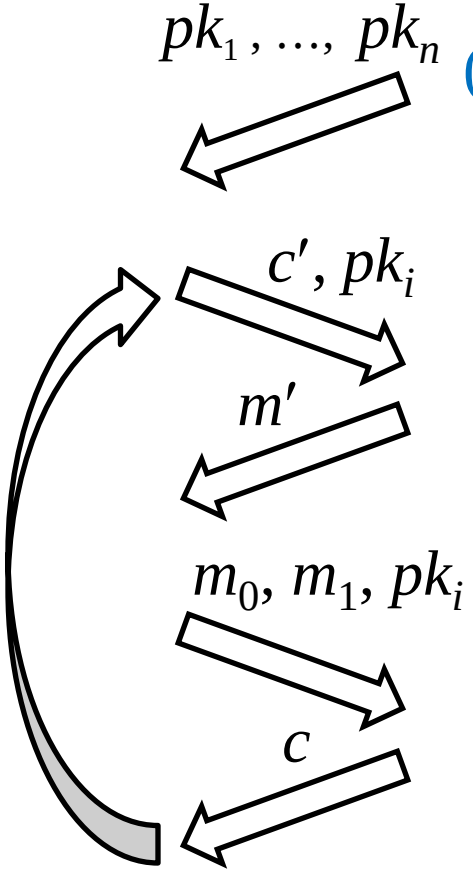
拡張された IND-CCA2 ゲーム

- 元締めは、多項式個の鍵の組を生成する。
- LR オラクルは、ビット b と二つのメッセージ (m_0,m_1) に加えて、公開鍵 pk を引数とする。隠された b に従って、m_b を pk で暗号化した結果 c を返す。
- 復号オラクルは、暗号文 c と公開鍵 pk に対して、c を pk の秘密鍵で復号した結果を返す。ただし、復号オラクルに LR オラクルが返した暗号文を送ることはできない。
- ハイブリッド論法により、もともとの IND-CCA2 ゲームに還元可能。

拡張された IND-CCA2 ゲーム

攻撃者

元締め



$(pk_1, sk_1), \ldots, (pk_n, sk_n)$ を生成

b をランダムに選ぶ

c' は、オラクルが
返した以外の暗号文

c' を sk_i で復号した
結果を m' とする

m_b を pk_i で暗号化した
結果を c とする

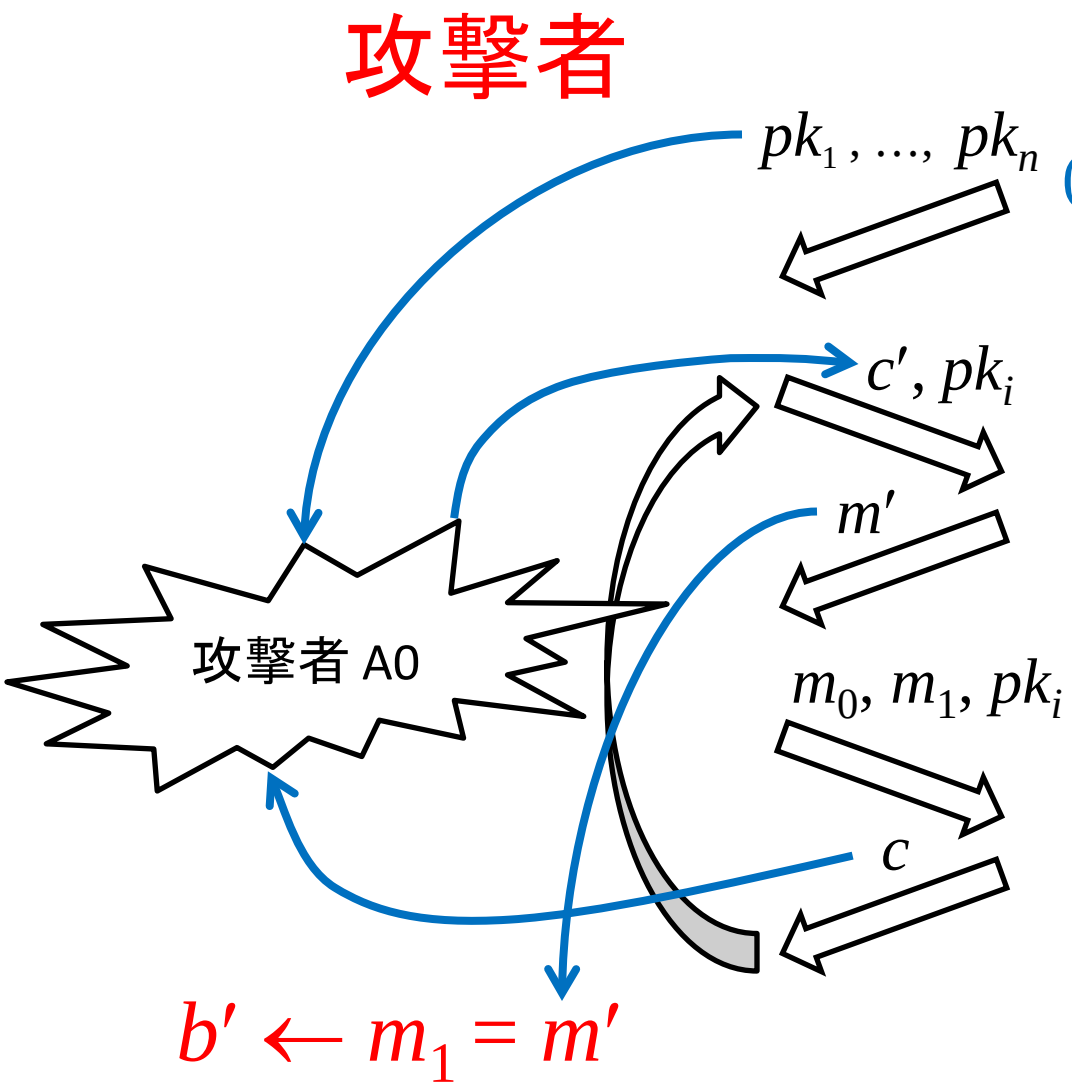
b の値を推測した
結果を b' とする

$b = b'$?

攻撃者 A

- A は $1..p(\eta)$ の間の数 k をランダムに選ぶ。
- A は、A0 に対してプロトコルのシミュレーションを行うが、k 個目のノンスもしくは暗号文 m に対して、二つのビット列 (m_0,m_1) を生成する。
  - ノンスの場合は、二つのランダムなビット列。
  - 暗号文の場合は、同じ平文を二回暗号化する。
  - 具体的には、アンパージングの際に生成する。
  - このノンスもしくは暗号文は、(さらに)暗号化されてネットワークに送信されることが期待される。
    - そうでないと Dolev-Yao に違反しない。

攻撃者 A


攻撃者
元締め
$pk_1, \ldots, pk_n$
$(pk_1, sk_1), \ldots, (pk_n, sk_n)$ を生成
b をランダムに選ぶ
c', pk_i
c' を sk_i で復号した
結果を m' とする
m'
攻撃者 A0
m_0, m_1, pk_i
m_b を pk_i で暗号化した
結果を c とする
c
$b' \leftarrow m_1 = m'$

攻撃者 A

- 以後、m のところには、m_0 または m_1 があると思ってプロトコルを実行する。
- アンパージングの際に、m を含むメッセージを暗号化するときは、m_0 を含むものと m_1 を含むものを作り、LR オラクルに暗号化させる。
- パージングとアンパージングの際に、LR オラクルが返した暗号文を復号する必要があるときは、復号オラクルを呼ばずに、LR オラクルに送ったメッセージを用いてシミュレーションを続ける。

たとえば

- N_A に対して二つのビット列 m_0 と m_1 を生成。
- $\{A, N_A\}_{KB}{}^{r1}$ に対しては、$\langle A, m_0 \rangle$ に対応するビット列と $\langle A, m_1 \rangle$ に対応するビット列を LR オラクルに送り、KB で暗号化させる。その結果を c とする。
- 他の参加者が c を復号する場合、復号オラクルを呼ばずに、$\langle A, m_0 \rangle$ または $\langle A, m_1 \rangle$ が得られたとしてシミュレーションを続ける。
- 以後も、N_A を含むメッセージを暗号化するときは、LR オラクルを呼ぶ。

攻撃者 A

- 攻撃者 A0 の送ったビット列をパージングするときは、m_0 もしくは m_1 が得られないかどうかを常に調べる。
  - どこで Dolev-Yao に違反するか、攻撃者 A にはわからない。

たとえば

- $\{\{N_A\}_{KA}{}^{r4}\}_{KB}{}^{r5}$ のようなメッセージをパージングしたならば、復号オラクルを繰り返し呼び出して、m_0 か m_1 を取り出せる。
- m_0 が得られたならば、b=0 と推測できる。

証明スケッチ

- 攻撃者 A0 が無視できない確率 $q(\eta)$ で Dolev-Yao に違反するならば、攻撃者 A は、少なくとも $q(\eta)/p(\eta)$ の確率で拡張された IND-CCA2 ゲームに勝つことができる。
- これは矛盾。

Mapping Lemma により

- 認証性などの性質は証明できる。
  - たとえば、計算論的なトレースの中に、レスポンダのスレッド t_c があったとする。
  - そのトレースに対応する記号論的なスレッド t_s も、レスポンダの（記号論的な）スレッドである。
  - 記号論的に認証性が成り立てば、t_s に対応するイニシエータのスレッド t_s' が存在する。
  - t_s' には、パージングとアンパージングにより、計算論的なスレッド t_c' が対応しているはず。
  - t_c' は、t_c に対応するイニシエータのスレッドである。

Mapping Lemma により

- したがって、記号論的に認証性が成り立てば、計算論的にも認証性が成り立つ。
- 個々のトレースに関する性質で、パージングとアンパージングで写り合う性質
  - たとえばノンスの秘密性
- しかし、匿名性などの性質は、Mapping Lemma によっては示せない。

  → Comon-Cortier（2008）