修 士 論 文

代数仕様言語CafeOBJによる セキュリティプロトコルの形式化

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

加藤 淳

2004年3月

修 士 論 文

代数仕様言語CafeOBJによる セキュリティプロトコルの形式化

主指導教員　二木厚吉 教授

審査委員主査　二木厚吉 教授
審査委員　落水浩一郎 教授
審査委員　大堀淳 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

210020 加藤 淳

提出年月: 2004 年 2 月

概 要

ネットワークを利用して安全な通信を行う技術として暗号がある．しかし，安全性が保証されている強固な暗号技術を用いても，その通信プロトコルに欠陥がある場合，安全な通信が行えるとはいえない．プロトコルの安全性を検証する方法として形式的に検証する手法が有効であると考えられている．

本研究では，代数仕様言語 CafeOBJ を用いてセキュリティプロトコルの仕様を形式的に記述し，その仕様の検証を行う．

検証は代数仕様言語 CafeOBJ で証明を記述し，CafeOBJ 文書に基づき証明譜を記述する．そして，証明譜を CafeOBJ 処理系で実行させ，証明譜の各部分が正しいことを検証する．例題として，共通鍵暗号を利用した認証プロトコルである Otway-Rees 認証プロトコルを用いる．

目 次

第1章　はじめに

近年，インターネットに代表される広域情報ネットワークの急速な普及および発展に伴い，社会生活において情報技術はなくてはならないものになっている．同時にネットワークセキュリティについての関心が高まっており，多くの通信プロトコルの考案が盛んに行われている．これらのプロトコルの中に情報を暗号化し，通信者間で秘密の通信を行うためのプロトコル，セキュリティプロトコルがある．セキュリティプロトコルはネットワーク上での安全な電子商取引や電子選挙等への適用が期待されている．

セキュリティプロトコルの安全性を保証するための手段の一つとして，暗号という技術がある．しかし，どんなに強固な暗号を用いたとしても，通信者間で期待されないような情報の受け渡しが行われる事態が発生した場合，そのプロトコルは安全とはいえず，欠陥のあるプロトコルとなってしまう．このような欠陥のあるプロトコルの運用は社会的，経済的な損失だけでなく，運用者自身の信用が失われてしまう可能性もある．そのため，セキュリティプロトコルが安全であることを保証することは非常に重要なことである．より大規模なネットワークシステムが構築されゆく昨今では，これらの欠陥を人間の直感やプロトコルの運用上で発見することは，非常に難しいことである．

一般に，ソフトウェアの開発は上流工程である仕様記述 (基本計画) から，外部設計，内部設計，プログラム設計，下流工程である最終的なプログラムコードの作成，テストまでの行程を経て行われる．仕様記述の段階で欠陥のあるシステムを開発してしまった場合，システム運用上での社会的·経済的損失はまのがれない．システムの規模が大規模になればその損失は計り知れないものになる可能性がある．仕様記述の段階で矛盾のない完璧な仕様を作成することは非常に重要なことである．

セキュリティプロトコルの安全性を検証する方法としては形式的に検証する手法が，よく知られたプロトコル，NSPK プロトコルの不具合が報告されたことで有効であると考えられており，これまでに多くの手法が考案されている．形式的に検証する手法は，システムの仕様作成から最終的なプログラムコードの作成までを一貫して数学的議論に基づいて行う手法で，厳密性·無矛盾性の点で優れており，システムの高信頼性に効果が高い．

本稿では，セキュリティプロトコルの形式化について説明し，その安全性検証について記述する．形式化には代数仕様言語 CafeOBJ を用いる．具体的には，セキュリティプロトコルを観測遷移機械 (OTS) でモデル化を行い，作成したモデル，および示したいことを CafeOBJ で記述する．作成した CafeOBJ 文書に基づき，証明を CafeOBJ で記述する (証明譜)．証明譜を CafeOBJ 処理系で実行させることで示したいことの検証を行う．

観測遷移機械はシステムがどのように振舞かを観察する．すなわち，対象となるシステ

ムを構成し，それに関連する値およびそれらがシステムの実行に伴いどのように変化するのかを観測することでシステムの振舞をモデル化する．

この形式化の手法に基づいて認証プロトコルの一つである，Otway-Rees 認証プロトコルを例題として取り上げ，セキュリティプロトコルの形式化および検証を行っていく．

第2章　セキュリティプロトコル

2.1　ネットワークの危険性

情報ネットワークの利用は今や社会生活になくてはならないものになった．電子商取引や電子選挙のように財政界を動かすシステムから，メールのやりとりのような個人の間の情報交換においてもネットワークは大いに利用されている．ネットワークの普及はその広がりが進むにつれ，安全性への関心が高まっている．現にウイルスによるシステム破壊や進入による攻撃のように企業に莫大な被害を与えるものから，顧客情報漏洩などのように個人に対しても被害を与えるような事件がニュースを飛び交っている．

ネットワークに対しての安全性は何もこういった攻撃に対しての防御策を講じることだけではない．欠陥のないシステムを構築してゆくことも重要な安全性である．システム自体に欠陥がある場合，すなわちシステムの利用社が意図しない動作をするようであれば，コスト的な被害だけでなく，社会的な信頼に関しても多大な被害を被ってしまうからである．

ネットワーク上に潜む主な脅威としては次のようなものがある [8, 9].

- 盗聴

  盗聴は通信内容を第三者に無断に傍受されることである．重要な情報を盗聴された場合，システムに対しての侵入や破壊といった行為だけでなく，莫大な被害や信用を落とすといった可能性がある．

  ネットワークにつながっている以上は誰にでも容易に盗聴をすることができる．また，データ解析ツール等を使用することによってより高度な盗聴も可能となる．

  暗証番号等の重要な情報をネットワーク上で扱う場合には暗号化する等の対策をとらなければならない．

- 侵入

  侵入は何者かがネットワークシステムに侵入することで，重要な機密情報が盗み出されたり，情報の改竄，破壊されるなどの不正が行われる脅威である．侵入の手口としては，システムの弱点をついた侵入から，パスワード管理が徹底していないモバイル環境からの侵入などがある．

  侵入を防ぐ手段としては，パスワード管理を徹底することやファイアウォールを導入する等がある．

- なりすまし

  なりすましは何者かがシステムの関係者になりすましてシステムを無断で使用することである．このほかにも，電子商取引等において，正当な通信を行うものににになりすまして，意図しない取引を行うなどがある．なりすましによる手口はパスワードの奪取もしくは盗聴が大半を含めている．こうした攻撃にはパスワードの厳重管理が効果的である．

- 改竄

  改竄は記録情報が不正に改変されることである．商取引等で，口座番号や振り込み金額などを転送経路上で改竄するなどの攻撃を行う．ネットワークによる通信を行うシステムは第三者に盗聴や改竄されやすい構造になっている．ネットワークを利用する際にはこのことを十分留意しておかなければならない．

- 破壊

  破壊は不正を行う第三者がネットワークシステムに侵入等を行い，故意にデータの消去，あるいは破壊を行うことである．システム関係者による人為的ミスによる事故も破壊に含まれる．また，自然災害や事故等による物理的損害も破壊の一つである．特に近年，ウイルス感染等による破壊の被害は増加している．

  重要なデータなどは定期的にバックアップしておくことで対策を講じることができる．

以上のネットワーク上の脅威のほかにもコンピュータウイルスや，デマ情報などによる風評被害などネットワークには多くの脅威にさらされている．これらの脅威による被害はコスト的な損害だけではなく，信用を落としてしまう場合もある．

本研究では主に盗聴，改竄，なりすましの 3 つに対しての対処が目的となっている．

2.2 セキュリティプロトコル

セキュリティプロトコルは危険なネットワーク上で安全な通信を行うための通信プロトコルである．安全な通信とは，前節で説明したネットワークに対する脅威に対しての対処が行われており，不正な行為が行われないことである．

セキュリティプロトコルは次のような性質があり [4]：

- 秘密性
- 信頼性

秘密性は正当なユーザ同士の秘密通信において，その通信に関係のない不正なユーザがその通信内容を知ることができないことを意味しており，信頼性とは意図したメッセージがプロトコルに従って正確に送信されたことを意味している．

安全な通信を行うための方法として暗号化による通信が考案され，これまで多くの暗号化技術が実現されている．暗号技術は単に情報を暗号化して盗聴されても内容が読まれないようにする，秘密通信を保証するだけではない．デジタル署名のように暗号文を復号できることによって，暗号文を受け取ったユーザが正当なユーザであることを証明する認証にも利用される．

セキュリティプロトコルはこの暗号技術を用いたものが多い．プロトコルにおいて，それに適した暗号技術を用いることによって，信頼のできる通信路を確立することができる．セキュリティプロトコルは認証，制御，防御の 3 つの要素からなっており，それぞれの役割は次のようになっている．

- 認証

  システムの正当なユーザであることを確認 · 判別を行う．判別できない場合は拒否する．パスワード[1]のように相手が基地のユーザであることが理解もしくは証明できればよい．主な認証方法としては，パスワードによる認証，指紋 · 網膜による認証，デジタル証明書などによる認証がある．

- 制御

  認証を元にユーザにサービスを割り当てる．また，ユーザ毎に適切なサービスを行うことを管理する．認証と制御の実装は一体であることが多い．

- 防御

  外部からの攻撃者に対して自らを防御する．防御項目を設定し，不正なサービスに対して防御を行う．主なものとしてファイアウォールがある．

あるネットワークシステムがユーザに対してサービスを提供する場合は次のような手順を踏む．まずシステムの正当な利用者であるかの認証を行う．認証により正当なユーザであることが確認できたら，そのユーザに対して適切なサービスを提供する．システムの運営中には外部からの侵入者等から攻撃を受けたり，予測できない事故が発生したときのためにシステムが無事であるように防御幕を張っておく．セキュリティプロトコルは以上のような手順を満たすことによって，安全なシステムを構築していく．

本研究では SSH や SSL などに代表される認証プロトコルに焦点を当てている．オープンなネットワーク上で通信者間がメッセージを暗号化することによって他者が参加できない秘密の通信を行うことができる．認証プロトコルはこの秘密通信を利用して認証を行う．

秘密通信および認証は電子商取引や電子選挙等への適用が期待されており，多くの方法が考案されている．

[1] 容易に類推できるようなものではならない．

2.3 NSLPK 認証プロトコル

セキュリティプロトコルの代表的な例として NSLPK 認証プロトコルがある．NSLPK 認証プロトコルは 1978 年，Needham と Schroeder により公開鍵暗号方式を用いた認証プロトコルとして提案された，NSPK プロトコルが元となっている．NSPK プロトコルは考案されてから 17 年後の 1995 年に Lowe により，侵入者が他の主体になりすまし別の主体との認証を確立させてしまうという欠陥が発見された [3]．この欠陥を発見と同時に Lowe は修正案を提案した．この修正案を NSLPK プロトコルと呼ぶ．

NSPK プロトコルの欠陥とは侵入者型の主体になりすまし別の主体との認証を確立できるというものである．具体的には次のようになっている．

$$
\begin{aligned}
Message1.\quad & A \to B \quad : \quad \{N_A, I_A\}_{K_B} \\
Message2.\quad & B \to A \quad : \quad \{N_A, N_B\}_{K_A} \\
Message3.\quad & A \to B \quad : \quad \{N_B, \}_{K_B}
\end{aligned}
$$

NSPK 認証プロトコル

上記のようなプロトコルにおいて，主体 C が不正をはたらく．

- STEP1　　$A \to C : \{N_A, A\}_{K_C}$

  主体 A は主体 C と通信を開始する．主体 A は主体 C に対して Message1 を送信する．

- STEP2　　$C \to B : \{N_A, A\}_{K_B}$

  Message1 を受信して，復号した主体 C は復号した Message1 をそのまま主体 B の公開鍵で暗号化し主体 B に送信する．

- STEP3　　$B \to C(A) : \{N_A, N_B\}_{K_A}$

  Message1 を受信し，復号した主体 B はメッセージ中に主体 A の識別子とノンスを確認し，メッセージ中ノンスと生成した自分のノンスを主体 A の公開鍵で暗号化し主体 C に送信する．ここで，主体 B はメッセージ中に主体 A の識別子が含まれていることから通信相手を A と判断する．主体 B が送信するメッセージは主体 C に横取りされるが，主体 A の公開鍵で暗号化されているため，メッセージを復号することはできない．

- STEP4　　$C \to A : \{N_A, N_B\}_{K_A}$

  主体 B からのメッセージを受け取った主体 C はメッセージには手を加えずそのまま主体 A に送信する．

- STEP5　　$A \to C : \{N_B\}_{K_C}$

  主体 $C(B)$ からのメッセージを受信した主体 A は暗号文を復号し，メッセージ中に Message1 で生成したノンスが含まれていることを確認する．ここで，通信を行っている主体 A にとっての通信相手は主体 C である．なぜなら，受け取ったメッセージに主体 C の公開鍵で暗号化したノンスが含まれているからである．したがって，主体 A は Message3 として Message2 の通信相手のノンスを主体 C の公開鍵で暗号化して送信する．この時点で，主体 A は主体 C の存在を確認して認証が確立する．

- STEP6　　$C(A) \to : \{N_B\}_{K_B}$

  Message3 を受信し，復号した主体 C は主体 A との認証が確立する．また，そのメッセージを主体 B の公開鍵で暗号化して送信する．暗号文を受け取った主体 B は自分の秘密鍵で復号し，自分が生成したノンスと一致することを確認する．これにより通信相手を主体 A と確認し認証する．

Lowe は Message2 に送信者の識別子を追加することでこの欠陥を解消した．

NSLPK 認証プロトコルは各主体に対して公開鍵が与えられる公開鍵暗号方式を用いている．公開鍵とは Otway-Rees 認証プロトコルで使用した共通鍵と違い，暗号化に使う鍵と複合化に使う鍵が別のものとなっている．そのうち片方を公開鍵として鍵サーバ等で管理し，広く世間に公開される．他の主体はこの共通鍵を自由に取得することができる．もう一方はユーザ自身が他に漏れないように管理しなければならない．

NSLPK 認証プロトコルでは Otway-Rees 認証プロトコル同様一つの値を一つの目的でしか使用しない，ノンスを用いる．ここでもノンスは類推不可能なものである．NSLPK 認証プロトコルは次のようなプロトコルとなっている．主体 A の公開鍵は K_A のように記述し，それに対する秘密鍵を K_A^{-1} と記述する．

$$
\begin{aligned}
Message1.\quad & A \to B \quad : \quad \{N_A, I_A\}_{K_B} \\
Message2.\quad & B \to A \quad : \quad \{N_A, N_B, I_B\}_{K_A} \\
Message3.\quad & A \to B \quad : \quad \{N_B,\}_{K_B}
\end{aligned}
$$

NSLPK 認証プロトコル

主体 A と主体 B が相互に認証をしたい場合，主体 A はノンス N_A を生成し，識別子 I_A とともに主体 B の公開鍵で暗号化した暗号文 $\{N_A, I_A\}_{K_B}$ を主体 B に送る．(Message1)

Message1 を受信した主体 B は暗号文を自分の秘密鍵 K_B^{-1} で復号し，ノンス N_A と，識別子 I_A を取得する．識別子が送信者の識別子であることを確認したら，ノンス N_B を生成する．受け取ったノンス N_A，と自身の識別子 I_B，と生成したノンス N_B を主体 A の公開鍵で暗号化する．暗号文 $\{N_A, N_B, I_B\}_K A$ を主体 A に送信する．(Message2)

Message2 を受信した主体 A は暗号文を自分の秘密鍵 K_A^{-1} で復号し，2 つのノンス N_A，N_B と識別子 I_B を得る．得られた識別子が送信者の識別子と同じことを確認する．続い

て，自分が生成したノンス N_A が一致することを確認する．このことより確かに通信相手が主体 B であったことを確認する．これは，ノンス N_B を取得できるのは秘密鍵 K_B^{-1} を所持している主体のみだけだからである．これを確認した後，ノンス N_B を主体 B の公開鍵 K_B で暗号化した暗号文 $\{N_B\}_{K_B}$ を主体 B に送る．(Message3)

Message3 を受信した主体 B は暗号文を自分の秘密鍵 K_B^{-1} で復号し，ノンス N_B を取得し，主体 A のために生成したノンスであるかの確認する．これより，通信相手が確かに主体 A であることを確認する．

2 つノンスを共有した主体 A，B はこのノンスを基にセッションキーを作成し，その鍵で暗号化することにより秘密の通信をすることができる．

第3章　代数仕様言語CafeOBJ

3.1　形式仕様

システムの仕様とは，その構成に対しての利用者の要求や，設計，振る舞い，問題，問題の解決策などを定義するものである．仕様は，真偽が客観的に決定できる記述であり，無矛盾性，完全性，非曖昧性，また，検証やテストが可能であること，さらにわかりやすさを持つことが望ましい

形式仕様とは，仕様記述の段階で，文法·意味を厳密な数学モデルや論理体系に基づいて定義された言語を用いて表現された仕様のことである．このことより，無矛盾性などのいろいろな性質を仕様のレベルで解析できる可能性が高く，より信頼性の高い仕様を作成することができる．また形式言語 (formal language) で記述されるため仕様自体の機械処理が可能であり，機械的検証も可能であるため，ソフトウェア開発の効率面でも多くの利点を持っている．

一方，自然言語，図，ダイアグラムなどを用いて書かれた仕様のように意味を定めるモデルや規則が与えられていない非形式な仕様では，可読性が高い反面，無矛盾性，完全性，非曖昧性などの性質を仕様が満たすかどうかを判断することが困難である．

信頼性が求められるようなシステム開発において形式仕様を採用することは非常に有効な手段である．

3.2　代数仕様言語CafeOBJ

代数仕様言語 CafeOBJ は主に始代数 (inital algebra) と隠蔽代数 (hidden algebra) に基づいている仕様記述言語である [2, 7]．始代数は主に自然数やスタックなどの抽象データ型の記述に，隠蔽代数は抽象機械の記述に用いる．CafeOBJ にはプログラミング言語の方に相当する 2 種類のソートがある．抽象データ型を表わす可視ソート (visible sort) と抽象機械の状態空間を表わす隠蔽ソート (hidden sort) である．隠蔽ソートについては，抽象機械の状態を変化させるのに用いる作用演算 (action operation) と抽象機械の状態を観察するのに用いる観測演算 (observatino operation) の 2 種類の演算がある．

作用演算は抽象機械の状態と 0 個以上のデータを引数にとり抽象機械の変化後の状態を返す，すなわち，1 つの隠蔽ソートと 0 個以上の可視ソートから隠蔽ソートへの演算である．観測演算は抽象機械の状態と 0 個以上のデータを引数にとりその状態に関する値を返す，すなわち，一つの隠蔽ソートと 0 個以上の可視ソートから可視ソートへの演算である．

可視ソートは'['と']'でかこって宣言する．隠蔽ソートは'*['と']*'でかこって宣言する．もしソートに順序がある場合には [Sort1 < Sort2] のように記述することにより，Sort1⊂Sort2 のような包含関係が成り立つ．作用演算と観測演算の宣言は'bop'で始め，そのほかの演算は'op'で宣言する．'bop','op'の後には，演算子を記述し，':'と引数のソート列を書く．最後に'->'と結果のソートを書く．また，演算の属性として，結合律や交換律，冪等律等を宣言することができ，演算子の最後にそれぞれ'assoc','comm','idem'を付加する．等式の宣言は'eq'で開始する．条件付等式の宣言は'ceq'で開始する．'eq'の後は左辺式と右辺式を'='で連結し，最後に' .'を付ける．'ceq'の後は左辺式と右辺式を'='で連結し，続いて'if'および条件式を記述し，最後に' .'を付ける．

rmCafeOBJ はモジュールという記述単位を取り，

```
module モジュール名{
```

で書き出しが始まり，対応する } でモジュールの記述が終わる．モジュールは signature と axioms の 2 つの部分からなる．signature 部は，ソートと演算子の宣言を行う指標で，axioms 部は等式および遷移規則の定義を行う．モジュールはパラメタ化でき，リスト等を記述することができる．モジュールには組み込みのモジュールや定義されたモジュールを輸入することが可能である．輸入されるモジュールが拡張される場合には extending を使用し，変更がない場合には protecting を使用する．

CafeOBJ の実装には CafeOBJ 処理系を用いる．CafeOBJ 処理系は記述された仕様の等式を左から右への書き換え規則とみなし，与えられた項を書き換えることで，実行可能となり，仕様記述の段階でシステムの模擬実験を行ったり，システムがある性質を有することを証明したりできる．

第4章　観測遷移機械

4.1　システムのモデル

本研究ではセキュリティプロトコルのモデル化に観測遷移機械 (Observational Transition System) を用いる [2, 7]．観測遷移機械では，対象システムに関連する値が状態遷移によりどのように変化するかを状態の外部から観察することでモデルを作成する．

モデル化の対象となるシステムの状態空間を包含する状態空間 Υ の存在を仮定する．状態空間の各要素は状態と呼ぶ．

観測遷移機械 $\mathcal{S}$ は 3 組の $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ で定義され，各要素は次のようになっている．

- $\mathcal{O}$：観測の集合

  観測の集合 $o \in \mathcal{O}$ は状態を引数にとり，自然数などのデータ型 D を返す関数 $o : \Upsilon \rightarrow D$ である．返戻値を観測値と呼ぶ．

- $\mathcal{I}$：初期状態の集合

  各観測の集合 $o \in \mathcal{O}$ について，その初期値を決める条件で，$\mathcal{I} \subset \Upsilon$ である．

- $\mathcal{T}$：条件付遷移規則の集合

  各遷移規則 $\tau \in \mathcal{T}$ は，ある状態から次の状態を返す関数である．τ は効力条件と事後状態の 2 組で定義する．効力条件は各遷移規則 $\tau \in \mathcal{T}$ に付随する条件 $c_\tau : \Upsilon \rightarrow \{true, false\}$ で表わす．各状態 $v \in \Upsilon$ に対して，$\tau(v)$ を v の τ に関する事後状態と呼ぶ．c_τ が真であるときは τ は効力があり，各観測 o は事後状態で定義した値に変化する．c_τ が偽であるときは各観測 o は変化しない．

観測遷移機械 $\mathcal{S}$ の実行は，初期状態からはじまり，実行の各段階で遷移規則の一つを非決定的に選択し適用することによって得られる状態の無限列である．非決定的選択とはすべての遷移規則が際限なく選択される公平性 (fairness) を満たすことである．

観測遷移機械 $\mathcal{S}$ の実行は，次の条件を満たす状態の無限列 $v_0, v_1, ...$ である．

- 開始性：

  状態 v_0 において，各観測 $o \in \mathcal{O}$ が $v_0 \in \mathcal{I}$ を満たす．

- 連続性：

  すべての $i \in \{0, 1, 2, ...\}$ に対して，$v_{i+1} = s\ \tau(v_i)$ を満たす $\tau \in \mathcal{T}$ が存在する．

- 公平性：

  各 $\tau \in \mathcal{T}$ に対し，$v_{i+1} = s\ \tau(v_i)$ を満たす $i \in \{0, 1, 2, ...\}$ が無限に存在する．

状態が $\mathcal{S}$ の実行に現われるとき，その状態は $\mathcal{S}$ に関して到達可能であるという．

4.2 検証

観測遷移機械 $\mathcal{S}$ にとって重要な性質は安全性 (safety property) と活性 (liveness property) である．本稿では安全性のみについて議論していく．$\mathcal{S}$ が安全性を有するとは，$\mathcal{S}$ のあらゆる実行において，その安全性を侵す危険な状態に陥らないことである．すなわち，$\mathcal{S}$ の任意の到達可能状態 $v \in \Upsilon$ において，$p(v)$ が真であることである．(p は述語 $p : \Upsilon \to \{true, false\}$)

$\mathcal{S}$ が安全性 p を有することを帰納法を用いて検証する．帰納法の手続きでは，次の遷移規則についての確認を行う．

- 基底：

  $v \in \mathcal{I}$ を満たす任意の初期状態 v において，$p(v)$ が成り立つ．

- 帰納段階：

  $p(v)$ が成り立つ任意の到達可能な状態 v において，すべての遷移規則 $\tau \in \mathcal{T}$ に対し，$p(\tau(v))$ が成り立つ．

4.3 CafeOBJ による観測遷移機械の記述

状態空間 Υ は H を隠蔽ソートとしたとき，`*[`H`]*`のように宣言することで表現する．可視ソート $V_k(k = i_1, ..., i_m)$ および V は各データ型 $D_k(k = i_1, ..., i_m)$ および D を始代数に基づいて定義することで表現する．

観測遷移機械 $\mathcal{S}$ の観測 $o_{i_1,...,i_m} \in \mathcal{O}$ は CafeOBJ の観測演算で表現する．観測 o に対応する観測演算は次のように宣言する．

`bop` o : $H\ V_{i_1}...V_{i_m}$ `->` V

初期条件 I（初期状態での各観測値）は初期状態を表わす定数を宣言し，各観測値を等式で定義する．初期状態を表わす定数を $init$ とし次のように宣言する．

`op` $init$: `->` H

ソート V_k の CafeOBJ 変数を $X_k(k = i_1, ..., i_m)$ とする．観測 $o_{i_1,...,i_m} \in \mathcal{O}$ の初期値が $f(i_1, ..., i_m)$ で表わされる関数で与えられると，

`eq` $o(init, X_{i_1}, ..., X_{i_m})$ `=` $f(X_{i_1}, ..., X_{i_m})$.

と記述できる．

観測遷移機械 S の遷移規則 $\tau_{j_1,...,j_n} \in \mathcal{T}$ は CafeOBJ の作用演算で表現する．遷移規則に対応する作用演算は次のように宣言する．

bop a : H $V_{j_i}...V_{j_n}$ -> H

状態 W をソート H の CafeOBJ 変数する．状態 W に対応する遷移規則を適用した後の事後状態を $a(W, X_{j_1}, ..., X_{j_n})$ と表わす．e-a$(W, X_{j_i}, ...X_{j_n}, X_{i_1}, ..., X_{i_m})$ を遷移規則が効力のある状態で実行された場合の観測 o の事後状態における観測値に対応する CafeOBJ の項とし，c-a$(W, X_{j_1}, ..., X_{j_n})$ を遷移規則の効力条件 $c_{\tau_{j_1,...,j_n}}$ に対応する CafeOBJ の項とする．

効力条件が真である状態における遷移規則 $\tau_{j_i,...,j_n}$ の実行に伴う，観測 $o_{i_1,...,i_m}$ の観測値の変化は次のように記述する．

ceq $o(a(W, X_{j_1}, ..., X_{j_n}), X_{i_1}, ..., X_{i_m})$
= $e - a(W, X_{j_i}, ...X_{j_n}, X_{i_1}, ..., X_{i_m})$
if $c - a(W, X_{j_1}, ..., X_{j_n})$.

効力条件が偽であればどの観測値も変化しないので次のように記述する．

ceq $a(W, X_{j_1}, ..., X_{j_n})$
= X if not $c - a(W, X_{j_1}, ..., X_{j_n})$.

効力条件の真偽にかかわらず観測値を変化させない場合は

eq $o(a(W, X_{j_1}, ..., X_{j_n}), X_{i_1}, ..., X_{i_m})$
= $o(S, X_{i_1}, ..., X_{i_m})$.

のように記述する．

4.4 検証の指針

観測遷移機械 $\mathcal{S}$ が安全性 p を有するとは，$\mathcal{S}$ に関して到達可能なすべての状態において p が成り立つことである．これより，遷移規則を帰納的に適用することにより検証することが可能となる．ただし，一般に p を単独で検証を試みようとすると帰納法の仮定が弱いため，帰納段階の証明が遂行できなくなる場合がある．この場合，別の補題を用いる必要がある．

4.4.1 検証したい性質の記述

検証を行う際はじめに，検証したい性質を CafeOBJ の項として記述する．まず，性質(述語)$p_k(k = 1, ..., n)$ を記述するモジュール INV を宣言する．このモジュールに各 p_k を表

わす演算子を次のように宣言する．

```
op p1 :  Sys SortX1 -> Bool
⋮
op pn :  Sys SortXn -> Bool
eq p1(W,X1) = P1(W,X1) .
⋮
eq pn(W,Xn) = Pn(W,Xn) .
```

ここで Sys は隠蔽ソートで，SortX$_1$ は p に含まれる状態を表わすものを除く自由変数に対応する可視ソートの列である．W はソート Sys の CafeOBJ 変数を表わし，X$_k$ はソート列 SortX$_k$ の CafeOBJ 変数を表わす．eq で宣言している等式は op で宣言した演算子を定義する等式の宣言である．モジュール INV には SortX$_k$ に含まれる可視ソートの任意の値を表わす定数を宣言する．

次に帰納段階において証明すべき述語を記述するモジュールを宣言する．

```
op istep1 : SortX1 -> Bool
⋮
op istepn : SortXn -> Bool
eq istep1(X1) = p1(s,X1) implies p1(s',X1) .
⋮
eq istepn(Xn) = pn(s,Xn) implies pn (s',Xn) .
```

s は任意の到達可能な状態を，s' は状態 s で遷移規則が適用された事後状態を表わす定数として宣言する．eq で宣言している等式は op で宣言した演算子を定義する等式である．_implies_は論理含意を表わす演算子である．

4.4.2 証明譜の記述

証明譜の記述ははじめに任意の初期状態で，証明したい性質が成り立つことを示すために次のように証明譜を記述する．

```
open INV
  red p1(inti,x1) .
close
```

open は指定したモジュールに宣言されているソート，演算子，等式を利用することを宣言

する CafeOBJ コマンドである．close はその利用を終了することを宣言するコマンドである．red は等式を左から右への書き換え規則と見なし，与えられた項を書き換える (簡約する)．

各帰納段階において CafeOBJ の作用演算 a で表わされる遷移規則が証明する性質を保持していることを示すとき，状態空間を複数に分割する (場合分け)．分割した各場合毎に次のように証明譜を記述する．

```
open ISTEP
  任意の値を表わす定数の宣言 .
  場合を表わす等式の宣言 .
  補題等からの事実を表わす等式の宣言 .
  eq s' = a(s,...)  .
  red istep_n(X_n) .
close
```

まず，作用演算 a で使われる任意の値を表わす定数，議論中の場合を表わす等式を宣言する．必要であれば補題と憂からじ実を表わす等式の宣言をする．eq で宣言された等式は，定数 s' が定数 s で表わされる状態で作用演算 a に対応する遷移規則が適用された事後状態を表わすことを意味する．最後に red コマンドで項を簡約する．

簡約した結果が期待通りであれば (この場合は true) この場合の証明がうまくいったことを意味する．うまくいかないときは，この場合の空間をさらに分割するか証明に補題を必要とするか判断する．議論中のプロトコルに検証したい性質を有していない可能性もある．

状態空間の場合分けは次のような手順で行う．

- 遷移規則の効力条件が成立する場合としない場合
  - 成立する場合
    状態空間をさらに分割する．
  - 成立しない場合
    遷移規則を適用しても変化しないので性質を保持するのは明らか．仕様の誤り等がないか確認するために検証をする．

第5章　Otway-Rees認証プロトコル

Otway-Rees プロトコルは，1987 年に D.Otway と O.Rees によって，共通鍵暗号方式を用いた認証プロトコルとして提案された [5]．このプロトコルの目的は認証局 S が 2 つの主体のために共通鍵を発行し分配することである．ネットワーク上には通信を行う 2 つの主体 A，主体 B と共通鍵を発行 · 分配を行う認証局 S が存在している．

共通鍵暗号は暗号化鍵と復号化鍵が同一である暗号で，暗号化して通信を行う 2 者が同じ鍵を共有している．X と Y との間の共通鍵を K_{XY} とし，これにより暗号化した文を $\{...\}_{K_{XY}}$ と書く．暗号文は共通鍵の持ち主 X，Y にしか復号することはできない．

Otway-Rees プロトコルでは各主体と認証局との認証のため，および一つのセッションが正しく終了することを確認するために 1 つの値を 1 つの目的でしか使わないノンス (nonce) を用いる．ノンスは類推可能なものとそうでないものに分類できる．類推可能なものは通し番号やタイムスタンプといったものによる．類推不可能なものは (疑似) 乱数等により実現できる．Otway-Rees プロトコルでは類推不可能なノンスを用いる．

各主体はあらかじめ認証局との共通鍵を共有していると仮定すると，Otway-Rees プロトコルの一つのセッションは次の 5 ステップで記述できる．

1. $A \to B:\ \ N, I_A, I_B, \{N, N_A, I_A, I_B\}_{K_{AS}}$
2. $B \to S:\ \ N, I_A, I_B, \{N, N_A, I_A, I_B\}_{K_{AS}}, \{N, N_B, I_A, I_B\}_{K_{BS}}$
3. $S \to B:\ \ N, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \to A:\ \ N, \{N_A, K_{AB}\}_{K_{AS}}$
5. $A \leftrightarrow B:\ \ \{anything\ recognizable\}_{K_{AB}}$

次に Otway-Rees プロトコルの手順を具体的に追う．

- STEP1

$$Message1 \quad A \to B:\ \ N, I_A, I_B, \{N, N_A, I_A, I_B\}_{K_{AS}}$$

主体 B と秘密の通信を行いたい主体 A は，秘密の通信を行うために必要な共通鍵を発行してもらうために 2 種類のノンス N, N_A を生成する．ノンス N はこれから行う

セッションを表すものである．ノンス N_A は主体 A が主体 B との共通鍵を発行してもらうために生成したもので，認証局 S との認証のために使う．このほかに必要な情報として主体 A と主体 B の識別子である I_A，I_B を用意する．

主体 A は 2 つのノンスと 2 つの識別子を主体 A と認証局との共通鍵 K_{AS} で暗号化する．これにノンス N，識別子 I_A，I_B を加えたものを主体 B に送信する (Message1)．

ここで注意することは，ノンス N_A は主体 A が認証局との認証のために使用するので第三者には決して漏れてはいけない情報であるということである．

- STEP2

$$Message2 \quad B \rightarrow S: \quad N, I_A, I_B, \{N, N_A, I_A, I_B\}_{K_{AS}}, \{N, N_B, I_A, I_B\}_{K_{BS}}$$

Message1 を受信した主体 B は主体 A 同様に認証局との認証を行うためにノンス N_B を生成する．次に Messagae1 中の暗号化されていない情報，ノンス N と識別子 I_A，I_B に生成したノンス N_B を加えて，主体 B と認証局との共通鍵 K_{BS} で暗号化を行う．生成した暗号文を Message1 に加えて認証局に送信する (Message2)．

- STEP3

$$Message3 \quad S \rightarrow B: \quad N, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$$

Message2 を受信した認証局 S は 2 つの暗号文を復号し，メッセージ中のノンス N，識別子 I_A，I_B が 2 つの暗号文中のノンス N，識別子 I_A，I_B とそれぞれ一致することを確認する．このことにより暗号文の生成者がそれぞれ主体 A，主体 B であることを確認する．これは，2 つの暗号文はメッセージ中に識別子で表されたそれぞれの主体との間の共通鍵でしか復号することができないからである．

これらを確認した後，認証局は主体 A と主体 B との共通鍵を発行する．発行した共通鍵は，主体 A が生成したノンス N_A と組み合わせ共通鍵 K_{AS} で暗号化，また主体 B が生成したノンス N_B と組み合わせ共通鍵 K_{BS} で暗号化する．

2 つの暗号文を生成した後ノンス N を加えて主体 B に送信する (Message3)．

- STEP4

$$Message4 \quad B \rightarrow A: \quad N, \{N_A, K_{AB}\}_{K_{AS}}$$

Message3 を受信した主体 B はメッセージ中のノンス N が自分が送った Message2 中のノンス N と一致することを確認する．また，自分と認証局との共通鍵 K_{BS} で暗号化された暗号文を復号し，自分が生成したノンス N_B が一致することを確認する．このことにより認証局との認証ができたことを確証する．

主体 B は，Message3 から暗号文 $\{...\}_{K_{BS}}$ を除いた残りを主体 A に送信する (Message4)．

- STEP5

$$Message5 \quad A \leftrightarrow B: \ \{anything\ recognizable\}_{K_{AB}}$$

Message4 を受信した主体 A はメッセージ中のノンス N が自分が生成したノンス N と一致すること確認する．併せて，暗号文を復号し，ノンス N_A が一致することを確認する．これにより認証局との認証および，鍵の分配が正しく行われたことを確認する．

これ以後，分配された共通鍵を使って主体 A と主体 B との秘密通信を行うことができる．

第6章　Otway-Rees認証プロトコルのモデル化

6.1　セキュリティプロトコルの仮定

システムを形式化し，シミュレーションを行う際にはそのシステムがどのような状況でどのようなことが出来るのか，どのような性質を持っているのかのように，あらかじめ仮定をしておく必要がある．本研究ではOtway-Rees認証プロトコルをモデル化するにあたって以下のような仮定をあらかじめ条件として定義する．

- プロトコルの参加者

  Otway-Rees認証プロトコルの参加者として，信頼できる主体，信頼できない主体，認証局の3種類を定義する．信頼できる主体とは，プロトコルに則った動作のみを行うプロトコルの正式な参加者である．信頼できない主体とは，プロトコルに則った動作に加えてネットワークを流れるメッセージを盗聴し，盗聴した情報を基にメッセージの偽造を行う．認証局はこのプロトコルでは唯一の存在で，プロトコルに則った動作のみを行い不正な行為は行わない．ただし，信頼できない主体によるなりすましは起こりうる．以後，信頼できる主体を単に主体と，信頼できない主体を侵入者として取り扱う．侵入者についての定義の詳細は後に記述する．

- 認証局との共通鍵

  侵入者も含めた各主体はあらかじめ認証局と共通鍵を共有している．Otway-Rees認証プロトコルの本来の目的は,「2つの主体が秘密の通信を行うために認証局に共通鍵を発行してもらう」というところにあるので，プロトコル中の暗号文の暗号鍵の受け渡しについては議論しない．

- 暗号の安全性

  本研究ではプロトコル自体に欠陥がないことを検証するので，利用する暗号は完全なものであり，第三者によって解読はできないものとする．従って暗号文中のノンスや共通鍵といった情報を復号すること以外で推測することはできない．

- ネットワークの危険性

  メッセージを媒介するネットワークは安全なものではない．一度ネットワークに入れられたメッセージは第三者によって盗聴することが可能である．

- セッション毎の鍵の生成

  モデル化するプロトコルでは，たとえ同じ主体同士の通信であっても，セッションが異なる場合は違う鍵を使用する．すなわち，セッション毎に認証局によって，共通鍵を発行してもらうとする．いわば，発行する共通鍵は使い捨てとする．これは，同じ鍵を使い続けることの危険に対しての考慮と，ユーザの鍵管理に対する負担の軽減のためである．同じ鍵を使い続けることはどんなに強固な暗号であっても使う時間が増していくと同時に，それだけ解読に時間をかけられることになり，非常に危険である．また，通信相手が増大していくことによって，それだけ鍵の管理をしなければならなくなる．

- 侵入者の定義

  侵入者は，ネットワークを流れるメッセージの盗聴等の，不正な動作を行い，プロトコルを攻撃する．一方で，侵入者はプロトコルの正式な参加者である可能性もある．そこで，侵入者は主体および認証局の一部として定義する．侵入者の行うことができることは以下のようになる．

  - ネットワークを流れるメッセージを盗聴する．
  - 盗聴したメッセージに含まれるデータを収集する．(暗号文については，侵入者と認証局との共通鍵で暗号化されている場合にのみ復号可能で，その情報を収集することができる.)
  - 収集したデータを基にメッセージを偽造し送信する．
  - 侵入者が共通鍵を発行することはできない．

  以上より，侵入者はネットワークからのみ情報を収集することができ，他の媒体から情報を収集することはできない．メッセージの偽造に関しては，利用可能なものは何でも利用し，何でも作ることができる．

6.2 モデル化で使用するデータ型の定義

Otway-Rees 認証プロトコルのモデル化で使用するデータ型を次のように CafeOBJ で定義していく．

モジュール PRINCIPAL は主体を定義する．指標 (signature) は以下のようになる．

```
[Principal]
op intruder : -> Principal
op _=_      : Principal Principal -> Bool {comm}
var P       : Principal
eq (P = P) = true .
```

Principalは主体のための可視ソートである．intruderは侵入者を主体の一部であり，汎用の侵入者としてモデル化している．Boolは論理式のための可視ソートである．演算子_=_は定義する各データ型に対する等しさを判定する．commは演算子_=_に与えられた属性で，演算が交換律を満たすことを宣言する．Pは可視ソートPrincipalのCafeOBJ変数であり，等式は主体の識別子の確認に用いる．

モジュールSERVERは認証局を定義する．

```
[Server]
op ca       : -> Server
op intruder : -> Server
op _=_      : Server Server -> Bool {comm}
var S       : Server
eq (S = S) = true .
```

Serverは認証局のための可視ソートである．caはプロトコルに唯一存在する認証局を定義している．intruderは侵入者を汎用の侵入者としてモデル化している．認証局はプロトコル中で唯一の存在ではあるが，侵入者によってメッセージの盗聴，偽造，送信が行われる場合があるので定義してある．ただし，侵入者によって共通鍵が発行されることはない．

モジュールMVALUE,NVALUEは2種類のノンスそれぞれに唯一性を持たせるための値を定義する．Mvalue,Nvalueはそれぞれ可視ソートを宣言している．

```
[Mvalue]
op _=_ : Mvalue Mvalue -> Bool {comm}
var MV : Mvalue
eq (MV = MV) = true .

[Nvalue]
op _=_ : Nvalue Nvalue -> Bool {comm}
var NV : Nvalue
eq (NV = NV) = true .
```

モジュールMNONCE,NONCEは2種類のノンスを定義する．ここで，2種類ノンスを別のソートとして定義しているのは，このプロトコルで使用する2種類のノンスのなす意味が大きく違うからである．MNONCEはある一つのセッションを表わし，セッションの違いを区別するために使用する．また，MNONCEはメッセージ中に暗号化されずに現れるため，誰にでも見ることができる．NONCEは主体が認証局との認証を行うために使用するもので，主体と認証局以外に見られてはならないもので，メッセージ中では暗号文の中に現れる．2種類のノンスの指標は次のようになる．

```
[Mnonce]
op mnonce  : Principal Principal Mvalue -> Mnonce
op creator : Mnonce -> Principal
op reciver : Mnonce -> Principal
op mv      : Mnonce -> Mvalue
op _=_     : Mnonce Mnonce -> Bool {Bool}

[Nonce]
op monce   : Principal Principal Nvalue -> Nonce
op creator : Nonce -> Principal
op reciver : Nonce -> Principal
op v       : Nonce -> Nvalue
op _=_     : Nonce Nonce -> Bool {Bool}
```

Mnonce,Nonceはそれぞれノンスのための可視ソートで，演算子mnonce,nonceはそれぞれノンスのデータの構成子である．最初の引数creatorはノンスを生成した主体を，2番目の引数reciverはどの主体と秘密の通信をするためのものかを表わしている．3番目の引数mv,vはノンスの唯一性を保証するための値を表わす．creator,reciverはメタな情報であり，ノンスを生成した主体および検証者以外これを利用することができない．すなわち，侵入者によってこの情報の偽造をできないことを表わす．項mnonce(p,q,mv)は主体pが主体qと秘密の通信をするために認証局にのために生成したノンスを表わし，mvはその唯一性を表わしている．

モジュールPUBKEYは認証局が発行する共通鍵の唯一性を持たせるための値を定義する．Pubkeyは可視ソートを宣言している．

```
[Pubkey]
op _=_ : Pubkey Pubkey -> Bool {comm}
var PK : Pubkey
eq (PK = PK) = true .
```

モジュールKEY,SKEYはそれぞれプロトコルで使用する共通鍵を定義する．KEYは認証局が発行する２つの主体のための共通鍵，SKEYはプロトコルであらかじめ共有している主体と認証局との共通鍵を表わしている．

```
[Key]
op key      : Principal Principal Pubkey -> Key
ops pr1 pr2 : Key -> Principal
op pk       : Key -> Pubkey
op _=_      : Key Key -> Bool {comm}
```

```
[Skey]
op skey : Principal -> Skey
op pr0  : Skey -> Principal
op _=_  : Skey Skey -> Bool {comm}
```

Key,Skey はそれぞれ可視ソートである．演算子 key の１番目と２番目の引数は認証局がどの主体とどの主体のために共通鍵を発行したかを表わしている．３番目の引数は発行した共通鍵の唯一性を保証するための値である．演算子 skey の引数は認証局との共通鍵を持っている主体を表わしている．認証局との共通鍵についてはプロトコル上でネットワークに流れることはないので唯一性を持たせる値を定義する必要がない．

モジュール CIPHER1 は Message1 および Message2 に現れる暗号文を定義する．

```
[Cipher1]
op enc1 : Skey Mnonce Nonce Principal Principal -> Cipher1
op skey : Cipher1 -> Skey
op mnonce : Cipher1 -> Mnonce
op nonce : Cipher1 -> Nonce
ops p q   : Cipher1 -> Principal
op _=_ : Cipher1 Cipher1 -> Bool {comm}
```

Cipher1 は Messag1，Message2 に現れる暗号文のための可視ソートである．演算子 enc1 は暗号文の構成子であり，最初の引数で暗号化および複合化に使用する共通鍵を表わしている．2 番目以降の引数は暗号文に現れる情報 (データ) を表わしており，2 番目の引数が一つのセッションを表わすノンス，3 番目の引数が認証局との認証を行うために使用するノンスを表わしており，4 番目，5 番目の引数は共通鍵を発行してもらう 2 つの主体を表わしている．暗号文 $\{N, N_A, I_A, I_B\}_{K_{AS}}$ はノンス N を項 m，ノンス N_A を項 n，主体 A,B を項 p,q と表わすと，項 enc1(sk(p),m,n,p,q) で表わされる．

モジュール CIPHER2 は Message3 および Message4 に現れる暗号文を定義する．

```
[Cipher2]
op enc2 : Skey Nonce Key -> Cipher2
op skey : Cipher2 -> Skey
op key : Cipher2 -> Mnonce
op nonce : Cipher2 -> Nonce
op _=_ : Cipher2 Cipher2 -> Bool {comm}
```

Cipher2 は Messag3，Message4 に現れる暗号文のための可視ソートである．演算子 enc2 は暗号文の構成子であり，最初の引数で暗号化および複合化に使用する共通鍵を表わしている．2 番目，3 番目の引数は暗号文に現れる情報 (データ) を表わしており，2 番目の引数が認証局との認証を行うために使用するノンスを表わしており，3 番目の引数が認証局

が発行した共通鍵を表わしている．暗号文 $\{N_A, K_{AB}\}_{K_{AS}}$ はノンス N_A を項 n，主体 A,B をそれぞれ項 p,q と表わすと，項 enc2(sk(p),n,k(p q)) で表わされる．

モジュール MSG はプロトコルに関連するメッセージ 4 種類を定義する．

```
[Msg]
op m1 : Principal Principal Principal Mnonce Principal Principal Cipher1 -> Msg
op m2 : Principal Principal Server    Mnonce Principal Principal
                                      Cipher1 Cipher1 -> Msg
op m3 : Server    Server    Principal Mnonce Cipher2   Cipher2 -> Msg
op m4 : Principal Principal Principal Mnonce Cipher2 -> Msg
--
ops m1? m2? m3? m4? : Msg -> Bool
ops creator1 sender1 reciver1 id1 id2 : Msg -> Principal
ops creator2 sender2 reciver2 : Msg -> Server
op mnonce : Msg -> Mnonce
ops ci1 ci2a ci2b : Msg -> Cipher1
ops ci3a ci3b ci4 : Msg -> Cipher2
op _=_ : Msg Msg -> Bool {comm}
```

Msg はメッセージのための可視ソートである．演算子 m1,m2,m3,m4 はそれぞれ，プロトコルの Message1 から Message4 に対応したデータ構成子である．4 つの構成子の最初の引数は各メッセージの作成者を，2 番目の引数が見かけ上のメッセージの送信者を，3 番目の引数が受信者をそれぞれ表わしている．4 番目以降の引数は各メッセージに含まれるデータを表わしている．ここで，メッセージの作成者と見かけ上の送信者を別の引数で表わしているのは，最初の引数はメタな情報でメッセージの作成者および外部の観察者以外には利用できないからである．すなわちこれは侵入者がこの値を偽造できないことを意味している．

演算子 m1?,m2?,m3?,m4?は，ネットワークに加えられたメッセージが Message1 から Message4 のどのメッセージなのかを判別するために用いる．残りの演算子は射影関数である．

モジュール EQTRIV は続く 3 つのパラメタ付モジュールの宣言に用いる．

パラメタ付モジュール BAG は汎用の多重集合を定義する．形式パラメタは D :: TRIV である．

```
[Elt.D < Bag]
op void  : -> Bag
op _,_   : Bag Bag -> Bag {assoc comm id: void}
op _\in_ : Elt.D Bag -> Bool
```

Bag は汎用の多重集合のための可視ソートである．ここで多重集合と定義しているのは Bag はネットワークを媒介するメッセージに関して利用するためである．ネットワーク上には侵入者が偽造した全く同じ情報が流れる可能性があるため多重集合としている．ここでは可視ソート Elt.D が可視ソート Bag より下位のソートであることを宣言している．従って，可視ソート Elt.D の項は可視ソート Bag の項としても扱うことが出来る．定数 void は空の多重集合を表わしている．演算子_,_は多重集合の構成子を表わしている．assoc は演算子が結合律を満たす属性を，id: void は定数 void が演算子の単位元である属性を演算子_,_に与える宣言をしている．演算子_\in_はある要素が多重集合に含まれているかを判断する述語である．

パラメタ付モジュール SET は汎用の集合を定義する．形式パラメタは D :: TRIV である．

```
[Elt.D < Set]
op empty : -> Set
op __    : Set Set -> Set {assoc comm idem id: empty}
op _\in_ : Elt.D Set -> Bool
```

Set は汎用の集合のための可視ソートである．Set はノンスや，暗号文，発行される共通鍵に利用するのでここでは多重集合ではなく，集合としている．可視ソート Elt.D は可視ソート Set より下位のソートであることを宣言している．従って，可視ソート Elt.D の項は可視ソート Set の項としても扱うことが出来る．定数 empty は空の多重集合を表わしている．演算子__は集合のデータ構成子である．idem は冪等律を満たす属性を演算子__に与える宣言をしている．演算子_\in_はある要素が集合に含まれているかを判断する述語である．

モジュール COLLECTION は汎用のものの集まりを定義する形式パラメタは D :: TRIV である．

```
[Elt.D < Col]
op _\in_ : Elt.D col -> Bool
```

Col は汎用のものの集まりのための可視ソートである．

モジュール NETWORK はネットワークを定義する．またモジュールではノンス，暗号文，共通鍵の侵入者がネットワークから取得できる情報を定義する．ネットワークはメッセージの多重集合としてモデル化する．一度ネットワークに入ったメッセージは侵入者により何度でも複製することが可能であることから，送信されネットワークに入れられたメッセージはネットワークから取り除かれることはない．従って，ネットワークが空である場合は，ネットワークに一度もメッセージが送信されていないことを表わしている．ここでは，パラメタ付モジュール BAG と COLLECTION を次のように具象化しモジュール内の可視ソートの名前を変更する．記号->の左辺に現われる可視ソート名を右辺の名前に変更する．

```
pr(BAG(MSG)*{sort Bag -> Network})
pr(COLLECTION(MNONCE)*{sort Col -> ColMnonce})
pr(COLLECTION(NONCE)*{sort Col -> ColNonce})
pr(COLLECTION(CIPHER1)*{sort Col -> ColCi1})
pr(COLLECTION(CIPHER2)*{sort Col -> ColCi2})
pr(COLLECTION(KEY)*{sort Col -> ColKey})
```

このモジュールの指標は次のようになる．

```
op cnonce : Network -> ColNonce
op cci1   : Network -> ColCi1
op cci2a  : Network -> ColCi1
op cci2b  : Network -> ColCi1
op cci3a  : Network -> ColCi2
op cci3b  : Network -> ColCi2
op cci4   : Network -> ColCi2
op ckey   : Network -> ColKey
```

演算子 cnonce は (図 6.1) 侵入者がネットワークから収集できる認証局との認証用のノンスを定義する．最初の等式はノンスの生成者が侵入者の場合そのノンスは利用できることを表わしている．2，3，4，5，6，7 番目の等式はネットワークにあるメッセージの暗号文が侵入者と認証局との共通鍵で暗号化されている場合，その暗号文中のノンスは利用できることを表わしている．最後の等式は上記のどの条件も満たさない場合はメッセージ内のノンスは利用できないことを定義する．

演算子 cci1 は (図 6.2) 侵入者がネットワークから収集できる Message1 中の暗号文を定義する．最初の等式はネットワークが空であれば侵入者が利用できる暗号文はないことを定義する．2 番目の等式はネットワーク中に Message1 が存在すれば，その中の暗号文を利用できることを定義する．ただし，暗号文が侵入者と認証局の共通鍵で暗号化されている場合は，侵入者がその暗号文を生成することはできるのでここでは除いている．最後の等式は Message1 以外のメッセージからは該当する暗号文を得られないことを定義する．

演算子 cci2a,cci2b,cci3a,cci3b,cci4 も同様に侵入者がネットワークから収集できる各メッセージ中の暗号文を定義している．

演算子 ckey は (図 6.3) 侵入者がネットワークから収集できる認証局が発行する認証鍵を定義する．最初の等式はネットワークが空であれば，侵入者が利用できる共通鍵はないことを定義する．2，3，4 番目の等式はネットワーク中の Message3 中の 2 つの暗号文のうちどちらかが侵入者と認証局の共通鍵で暗号化されている，および Message4 中の暗号文が侵入者と認証局の共通鍵で暗号化されていれば，暗号文中の共通鍵を利用できることを定義する．最後の等式はどの条件も満たさない場合はメッセージ内の共通鍵は利用できないことを定義する．

```
-- cnonce
eq N \in cnonce(void) = (creator(N) = intruder) .
ceq N \in cnonce(M,NW) = true
    if m1?(M) and pr0(skey(ci1(M))) = intruder .
ceq N \in cnonce(M,NW) = true
    if m2?(M) and pr0(skey(ci2a(M))) = intruder .
ceq N \in cnonce(M,NW) = true
    if m2?(M) and pr0(skey(ci2b(M))) = intruder .
ceq N \in cnonce(M,NW) = true
    if m3?(M) and pr0(skey(ci3a(M))) = intruder .
ceq N \in cnonce(M,NW) = true
    if m3?(M) and pr0(skey(ci3b(M))) = intruder .
ceq N \in cnonce(M,NW) = true
    if m4?(M) and pr0(skey(ci4(M))) = intruder .
ceq N \in cnonce(M,NW) = N \in cnonce(NW)
    if not(m1?(M) and pr0(skey(ci1(M))) = intruder) and
       not(m2?(M) and pr0(skey(ci2a(M))) = intruder) and
       not(m2?(M) and pr0(skey(ci2b(M))) = intruder) and
       not(m3?(M) and pr0(skey(ci3a(M))) = intruder) and
       not(m3?(M) and pr0(skey(ci3b(M))) = intruder) and
       not(m4?(M) and pr0(skey(ci4(M))) = intruder) .
```

図 6.1　侵入者がネットワークより利用できるノンスの定義

```
-- cci1
eq E1 \in cci1(void) = false .
ceq E1 \in cci1(M,NW) = true
    if m1?(M) and not(pr0(skey(ci1(M))) = intruder) and ci1(M) = E1 .
ceq E1 \in cci1(M,NW) = E1 \in cci1(NW)
    if not(m1?(M) and not(pr0(skey(ci1(M))) = intruder) and ci1(M) = E1) .
-- cci2a
eq E21 \in cci2a(void) = false .
ceq E21 \in cci2a(M,NW) = true
    if m2?(M) and not(pr0(skey(ci2a(M))) = intruder) and ci2a(M) = E21 .
ceq E21 \in cci2a(M,NW) = E21 \in cci2a(NW)
    if not(m2?(M) and not(pr0(skey(ci2a(M))) = intruder) and ci2a(M) = E21) .
-- cci2b
eq E22 \in cci2b(void) = false .
ceq E22 \in cci2b(M,NW) = true
    if m2?(M) and not(pr0(skey(ci2b(M))) = intruder) and ci2b(M) = E22 .
ceq E22 \in cci2b(M,NW) = E22 \in cci2b(NW)
    if not(m2?(M) and not(pr0(skey(ci2b(M))) = intruder) and ci2b(M) = E22) .
-- cci3a
eq E31 \in cci3a(void) = false .
ceq E31 \in cci3a(M,NW) = true
    if m3?(M) and not(pr0(skey(ci3a(M))) = intruder) and ci3a(M) = E31 .
ceq E31 \in cci3a(M,NW) = E31 \in cci3a(NW)
    if not(m3?(M) and not(pr0(skey(ci3a(M))) = intruder) and ci3a(M) = E31) .
-- cci3b
eq E32 \in cci3b(void) = false .
ceq E32 \in cci3b(M,NW) = true
    if m3?(M) and not(pr0(skey(ci3b(M))) = intruder) and ci3b(M) = E32 .
ceq E32 \in cci3b(M,NW) = E32 \in cci3b(NW)
    if not(m3?(M) and not(pr0(skey(ci3b(M))) = intruder) and ci3b(M) = E32) .
-- cci4
eq E4 \in cci4(void) = false .
ceq E4 \in cci4(M,NW) = true
    if m4?(M) and not(pr0(skey(ci4(M))) = intruder) and ci4(M) = E4 .
ceq E4 \in cci4(M,NW) = E4 \in cci4(NW)
    if not(m4?(M) and not(pr0(skey(ci4(M))) = intruder) and ci4(M) = E4) .
```

図 6.2　侵入者がネットワークより利用できる暗号文の定義

```
-- ckey 侵入者がネットワークから収集できる共通鍵
eq K \in ckey(void) = false .
ceq K \in ckey(M,NW) = true
    if m3?(M) and pr0(skey(ci3a(M))) = intruder
              and key(ci3a(M)) = K .
ceq K \in ckey(M,NW) = true
    if m3?(M) and pr0(skey(ci3b(M))) = intruder
              and key(ci3b(M)) = K .
ceq K \in ckey(M,NW) = true
    if m4?(M) and pr0(skey(ci4(M))) = intruder
              and key(ci4(M)) = K .
ceq K \in ckey(M,NW) = K \in ckey(NW)
    if not(m3?(M) and pr0(skey(ci3a(M))) = intruder
                  and key(ci3a(M)) = K) and
       not(m3?(M) and pr0(skey(ci3b(M))) = intruder
                  and key(ci3b(M)) = K) and
       not(m4?(M) and pr0(skey(ci4(M))) = intruder
                  and key(ci4(M)) = K) .
```

図 6.3　侵入者がネットワークより利用できる共通鍵の定義

6.3 Otway-Rees 認証プロトコルのモデル

Otway-Rees Protocol のモデルでは 4 つの観測と 16 の遷移規則を用いる．モジュール OTWAY-REES ではモデル化する観測遷移機械を記述する．パラメタモジュール SET は以下のように具象化して用いる．

```
SET(MVALUE)*{sort Set -> Umvalue}
SET(NVALUE)*{sort Set -> Uvalue}
SET(PUBKEY)*{sort Set -> Sharekey}
```

Umvalue,Uvalue,Sharekey はそれぞれプロトコルの認証子であるノンス，認証局との認証用のノンス，共通鍵の唯一性を持たせるための値のソートである．Otway-Rees モジュールの指標は次のようになる．

```
*[System]*
-- any initial state
op init : -> System

-- observation operations
bop uv : System -> Uvalue
bop um : System -> Umvalue
bop nw : System -> Network
bop sk : System -> Sharekey

-- action operations
-- sending messges
bop mes1 : System Principal Principal Mvalue Nvalue -> System
bop mes2 : System Principal Nvalue Msg  -> System
bop mes3 : System Msg Pubkey -> System
bop mes4 : System Principal Msg Msg Msg -> System

-- faking messages
-- for message1
bop fkm11 : System Principal Principal Mnonce Cipher1 -> System
bop fkm12 : System Principal Principal Mnonce Nonce Skey -> System
-- for message2
bop fkm21 : System Principal Principal Mnonce Cipher1 Cipher1 -> System
bop fkm22 : System Principal Principal Mnonce Cipher1 Nonce Skey -> System
bop fkm23 : System Principal Principal Mnonce Cipher1 Nonce Skey -> System
bop fkm24 : System Principal Principal Mnonce Nonce Nonce Skey Skey -> System
```

```
-- for message3
bop fkm31 : System Principal Principal Mnonce Cipher2 Cipher2 -> System
bop fkm32 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> System
bop fkm33 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> System
bop fkm34 : System Principal Principal Mnonce Nonce Nonce Skey Skey Key -> System
-- for message4
bop fkm41 : System Principal Principal Mnonce Cipher2 -> System
bop fkm42 : System Principal Principal Mnonce Nonce Skey Key -> System
```

隠蔽ソート System は状態空間 Υ に対応している．定数 init は任意の初期状態を表わしている．

観測演算 uv はプロトコルでそれまでに使用した認証局との認証用のノンスの集合を観察する．観測演算 um はプロトコルでそれまでに使用したプロトコルの識別子であるノンスの集合を観察する．観測演算 nw はプロトコルでそれまでに使用したメッセージを媒介するネットワークを観察する．観測演算 sk はプロトコルでセッション毎に生成される共通鍵の集合を観察する．観測演算 uv,um,sk の初期状態 init における観測値は empty で，観測演算 nw の初期状態 init における観測値は void で定義する．

作用演算 mes1,mes2,mes3,mes4 は，プロトコルに則った Message1 から Message4 までを送信する遷移規則に対応する．作用演算 fkm11,fkm12 は侵入者が Message1 を偽造する遷移規則に対応する．作用演算 fkm21,fkm22,fkm23,fkm24 は侵入者が Message2 を偽造する遷移規則に対応する．作用演算 fkm31,fkm32,fkm33,fkm34 は侵入者が Message3 を偽装する遷移規則に対応する．作用演算 fkm41,fkm42 は侵入者が Message4 を偽造する遷移規則に対応する．

作用演算 mes1 の 2 番目の引数はメッセージの作成，送信者を表わしており，3 番目の引数はメッセージの受信者を表わしている．(図 6.4) 4 番目の引数はプロトコル識別子であるノンス，5 番目の引数は 2 番目の引数で表わされる主体が認証局と認証を行うために生成したノンスを表わしている．

作用演算 mes2 の 3 番目の引数は，2 番目の引数で表わされる主体が認証局との認証のために生成したノンスを表わしており，4 番目の引数は 2 番目の引数で表わされる主体が受け取ったメッセージを表わしている．(図 6.4)

作用演算 mes3 の 2 番目の引数は認証局が受け取ったメッセージを表わしている．3 番目の引数は認証局が発行する共通鍵を表わしている．(図 6.5)

作用演算 mes4 の 3，4，5 番目の引数はそれぞれプロトコルの Message1，Message2，Message3 を表わしている．(図 6.5) 各メッセージを送信する効力条件と送信後の事後状態は表 6.1〜表 6.4 のようになっている．

作用演算 fkm11 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$ を侵入者が取得していることで，事後状態では偽造した Message1 をネットワークに追加する．

作用演算 fkm12 の効力条件はメッセージを偽造するのに必要なノンス N_A および共通鍵 K_{AS} を侵入者が取得していることで，事後状態では偽造した Message1 をネットワークに追加する．

作用演算 fkm21 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$，$\{...\}_{K_{BS}}$ を侵入者が取得していることで，事後状態では偽造した Message2 をネットワークに追加する．

作用演算 fkm22 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$，ノンス N_B および共通鍵 K_{BS} を侵入者が取得していることで，事後状態では偽造した Message2 をネットワークに追加する．

作用演算 fkm23 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{BS}}$，ノンス N_A および共通鍵 K_{AS} を侵入者が取得していることで，事後状態では偽造した Message2 をネットワークに追加する．

作用演算 fkm24 の効力条件はメッセージを偽造するのに必要なノンス N_A，N_B および共通鍵 K_{AS}，K_{BS} を侵入者が取得していることで，事後状態では偽造した Message2 をネットワークに追加する．

作用演算 fkm31 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$，$\{...\}_{K_{BS}}$ を侵入者が取得していることで，事後状態では偽造した Message3 をネットワークに追加する．

作用演算 fkm32 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$，ノンス N_B および共通鍵 K_{AS}，K_{AB} を侵入者が取得していることで，事後状態では偽造した Message3 をネットワークに追加する．

作用演算 fkm33 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{BS}}$，ノンス N_A および共通鍵 K_{AS}，K_{AB} を侵入者が取得していることで，事後状態では偽造した Message3 をネットワークに追加する．

作用演算 fkm34 の効力条件はメッセージを偽造するのに必要なノンス N_A，N_B および共通鍵 K_{AS}，K_{BS}，K_{AB} を侵入者が取得していることで，事後状態では偽造した Message3 をネットワークに追加する．

作用演算 fkm41 の効力条件はメッセージを偽造するのに必要な暗号文 $\{...\}_{K_{AS}}$ を侵入者が取得していることで，事後状態では偽造した Message4 をネットワークに追加する．

作用演算 fkm42 の効力条件はメッセージを偽造するのに必要なノンス N_A および共通鍵 K_{AS}，K_{AB} を侵入者が取得していることで事後状態では偽造した Message4 をネットワークに追加する．

表 6.1: Message1 を送信する条件と事後状態

効力条件	生成するノンス N, N_A が未使用
事後状態	観測演算 `uv` で観測されるノンス N_A を追加 観測演算 `um` で観測されるノンス N を追加 ネットワーク `nw` に Message1 を追加する

表 6.2: Message2 を送信する条件と事後状態

効力条件	生成するノンス N_B が未使用 次の条件を満たす Message1 が存在する ・受信者が 2 番目の引数で表わされる主体
事後状態	観測演算 `uv` で観測されるノンス N_B を追加 ネットワーク `nw` に Message2 を追加する

表 6.3: Message3 を送信する条件と事後状態

効力条件	発行する共通鍵 K_{AB} が未使用 次の条件を満たす Message2 が存在する ・受信者が認証局 ・メッセージ中に暗号鍵 K_{AS} で暗号化された暗号文が存在する ・メッセージ中に暗号鍵 K_{BS} で暗号化された暗号文が存在する ・メッセージ中のノンス N が 2 つの暗号文中のノンス N と一致する ・メッセージ中の識別子 I_A, I_B が 2 つの暗号文中の識別子 I_A, I_B と一致する
事後状態	観測演算 `sk` で観測される共通鍵 K_{AB} を追加 ネットワーク `nw` に Message3 を追加する

表 6.4: Message4 を送信する条件と事後状態

効力条件	生成するノンス N, N_A が未使用 次の条件を満たす Message1 が存在する ・受信者が 2 番目の引数で表わされる主体 次の条件を満たす Message2 が存在する ・見かけの送信者がが 2 番目の引数で表わされる主体 ・受信者が認証局 ・メッセージ中に暗号鍵 K_{BS} で暗号化された暗号文が存在する ・暗号文 $\{...\}_{K_{BS}}$ 中のノンス N がメッセージ中のノンス N と一致する ・暗号文 $\{...\}_{K_{BS}}$ 中の識別子 I_A, I_B がメッセージ中の識別子 I_A, I_B と一致する ・メッセージ中のノンス N が Message1 中のノンス N と一致する 次の条件を満たす Message3 が存在する ・見かけの送信者が認証局 ・受信者が 2 番目の引数で表わされる主体 ・メッセージ中に暗号鍵 K_{BS} で暗号化された暗号文が存在する ・メッセージ中のノンス N が Message2 中のノンス N と一致する ・暗号文 $\{...\}_{K_{BS}}$ 中のノンス N_B が Message2 で生成したノンス N_B と一致する
事後状態	ネットワーク `nw` に Message4 を追加する

```
-- for mes1
op c-mes1 :  System Principal Principal Mvalue Nvalue -> Bool
eq c-mes1(S,P,Q,MV,V1) = not(MV \in um(S)) and not(V1 \in uv(S)) .
ceq uv(mes1(S,P,Q,MV,V1)) = V1 uv(S) if c-mes1(S,P,Q,MV,V1) .
ceq um(mes1(S,P,Q,MV,V1)) = MV um(S) if c-mes1(S,P,Q,MV,V1) .
ceq nw(mes1(S,P,Q,MV,V1))
    = m1(P,P,Q,mnonce(P,Q,MV),P,Q,
         enc1(skey(P),mnonce(P,Q,MV),nonce(P,Q,V1),P,Q)) , nw(S)
      if c-mes1(S,P,Q,MV,V1) .
eq sk(mes1(S,P,Q,MV,V1)) = sk(S) .
ceq mes1(S,P,Q,MV,V1) = S if not c-mes1(S,P,Q,MV,V1) .

-- for mes2
op c-mes2 :  System Principal Nvalue Msg -> Bool
eq c-mes2(S,Q,V2,M1) = M1 \in nw(S) and m1?(M1) and
                       reciver1(M1) = Q and not(V2 \in uv(S)) .
eq um(mes2(S,Q,V2,M1)) = um(S) .
ceq uv(mes2(S,Q,V2,M1)) = V2 uv(S) if c-mes2(S,Q,V2,M1) .
ceq nw(mes2(S,Q,V2,M1))
    = m2(Q,Q,ca,mnonce(M1),id1(M1),Q,ci1(M1),
         enc1(skey(Q),mnonce(M1),nonce(Q,id1(M1),V2),id1(M1),id2(M1))) , nw(S)
      if c-mes2(S,Q,V2,M1) .
eq sk(mes2(S,Q,V2,M1)) = sk(S) .
ceq mes2(S,Q,V2,M1) = S if not c-mes2(S,Q,V2,M1) .
```

図 6.4　Otway-Rees 認証プロトコルの正規の Message1，Message2 を送信する定義式

```
-- for mes3
op c-mes3 :  System Msg Pubkey -⟩ Bool
eq c-mes3(S,M2,PK) = M2 \in nw(S) and m2?(M2) and reciver2(M2) = ca and
                     pr0(skey(ci2a(M2))) = id1(M2) and
                     pr0(skey(ci2b(M2))) = id2(M2) and
                     mnonce(M2) = mnonce(ci2a(M2)) and
                     mnonce(M2) = mnonce(ci2b(M2)) and
                     mnonce(ci2a(M2)) = mnonce(ci2b(M2)) and
                     p(ci2a(M2)) = id1(M2) and q(ci2a(M2)) = id2(M2) and
                     p(ci2b(M2)) = id1(M2) and q(ci2b(M2)) = id2(M2) and
                     not(PK \in sk(S)) .
eq um(mes3(S,M2,PK)) = um(S) .
eq uv(mes3(S,M2,PK)) = uv(S) .
ceq nw(mes3(S,M2,PK))
    = m3(reciver2(M2),reciver2(M2),sender1(M2),mnonce(M2),
         enc2(skey(id1(M2)),nonce(ci2a(M2)),key(id1(M2),id2(M2),PK)),
         enc2(skey(id2(M2)),nonce(ci2b(M2)),key(id1(M2),id2(M2),PK))) , nw(S)
      if c-mes3(S,M2,PK) .
ceq sk(mes3(S,M2,PK)) = PK sk(S) if c-mes3(S,M2,PK) .
ceq mes3(S,M2,PK) = S if not c-mes3(S,M2,PK) .

-- for mes4
op c-mes4 :  System Principal Msg Msg Msg -⟩ Bool
eq c-mes4(S,Q,M1,M2,M3) = M1 \in nw(S) and m1?(M1) and
                          M2 \in nw(S) and m2?(M2) and
                          M3 \in nw(S) and m3?(M3) and
                          reciver1(M1) = Q and sender1(M2) = Q and
                          reciver2(M2) = ca and pr0(skey(ci2b(M2))) = Q and
                          mnonce(M1) = mnonce(M2) and
                          mnonce(M2) = mnonce(ci2b(M2)) and
                          id2(M2) = Q and p(ci2b(M2)) = id1(M2) and
                          q(ci2b(M2)) = id2(M2) and sender2(M3) = ca and
                          reciver1(M3) = Q and pr0(skey(ci3b(M3))) = Q and
                          mnonce(M3) = mnonce(M2) and
                          nonce(ci3b(M3)) = nonce(ci2b(M2)) .
eq um(mes4(S,Q,M1,M2,M3)) = um(S) .
eq uv(mes4(S,Q,M1,M2,M3)) = uv(S) .
ceq nw(mes4(S,Q,M1,M2,M3))
    = m4(Q,Q,sender1(M1),mnonce(M3),ci3a(M3)) , nw(S)
      if c-mes4(S,Q,M1,M2,M3) .
eq sk(mes4(S,Q,M1,M2,M3)) = sk(S) .
ceq mes4(S,Q,M1,M2,M3) = S if not c-mes4(S,Q,M1,M2,M3) .
```

図6.5　Otway-Rees 認証プロトコルの正規の Message3，Message4 を送信する定義式

```
-- for action fkm11
op c-fkm11 :  System Principal Principal Mnonce Cipher1 -> Bool
eq c-fkm11(S,P,Q,MN,E11) = E11 \in cci1(nw(S)) .
eq uv(fkm11(S,P,Q,MN,E11)) = uv(S) .
ceq nw(fkm11(S,P,Q,MN,E11)) = m1(intruder,P,Q,MN,P,Q,E11) , nw(S)
                              if c-fkm11(S,P,Q,MN,E11) .
eq sk(fkm11(S,P,Q,MN,E11)) = sk(S) .
ceq fkm11(S,P,Q,MN,E11) = S if not c-fkm11(S,P,Q,MN,E11) .
-- for action fkm12
op c-fkm12 :  System Principal Principal Mnonce Nonce Skey -> Bool
eq c-fkm12(S,P,Q,MN,N1,K1) = N1 \in cnonce(nw(S)) and pr0(K1) = intruder .
eq uv(fkm12(S,P,Q,MN,N1,K1)) = uv(S) .
ceq nw(fkm12(S,P,Q,MN,N1,K1)) = m1(intruder,P,Q,MN,P,Q,enc1(K1,MN,N1,P,Q)) , nw(S)
                                if c-fkm12(S,P,Q,MN,N1,K1) .
eq sk(fkm12(S,P,Q,MN,N1,K1)) = sk(S) .
ceq fkm12(S,P,Q,MN,N1,K1) = S if not(c-fkm12(S,P,Q,MN,N1,K1)) .

-- for action fkm21
op c-fkm21 :  System Principal Principal Mnonce Cipher1 Cipher1 -> Bool
eq c-fkm21(S,P,Q,MN,E11,E12) = E11 \in cci2a(nw(S)) and E12 \in cci2b(nw(S)) .
eq uv(fkm21(S,P,Q,MN,E11,E12)) = uv(S) .
ceq nw(fkm21(S,P,Q,MN,E11,E12)) = m2(intruder,P,ca,MN,P,Q,E11,E12) , nw(S)
                                  if c-fkm21(S,P,Q,MN,E11,E12) .
eq sk(fkm21(S,P,Q,MN,E11,E12)) = sk(S) .
ceq fkm21(S,P,Q,MN,E11,E12) = S if not c-fkm21(S,P,Q,MN,E11,E12) .
-- for action fkm22
op c-fkm22 :  System Principal Principal Mnonce Cipher1 Nonce Skey -> Bool
eq c-fkm22(S,P,Q,MN,E11,N2,K2) = E11 \in cci2a(nw(S)) and
                                 N2 \in cnonce(nw(S)) and pr0(K2) = intruder .
eq uv(fkm22(S,P,Q,MN,E11,N2,K2)) = uv(S) .
ceq nw(fkm22(S,P,Q,MN,E11,N2,K2)) = m2(intruder,Q,ca,MN,P,Q,E11,enc1(K2,MN,N2,P,Q)) , nw(S)
                                    if c-fkm22(S,P,Q,MN,E11,N2,K2) .
eq sk(fkm22(S,P,Q,MN,E11,N2,K2)) = sk(S) .
ceq fkm22(S,P,Q,MN,E11,N2,K2) = S if not c-fkm22(S,P,Q,MN,E11,N2,K2) .
-- for action fkm23
op c-fkm23 :  System Principal Principal Mnonce Cipher1 Nonce Skey -> Bool
eq c-fkm23(S,P,Q,MN,E12,N1,K1) = E12 \in cci2b(nw(S)) and N1 \in cnonce(nw(S)) and
                                 pr0(K1) = intruder .
eq uv(fkm23(S,P,Q,MN,E12,N1,K1)) = uv(S) .
ceq nw(fkm23(S,P,Q,MN,E12,N1,K1)) = m2(intruder,Q,ca,MN,P,Q,enc1(K1,MN,N1,P,Q),E12) , nw(S)
                                    if c-fkm23(S,P,Q,MN,E12,N1,K1) .
eq sk(fkm23(S,P,Q,MN,E12,N1,K1)) = sk(S) .
ceq fkm23(S,P,Q,MN,E12,N1,K1) = S if not c-fkm23(S,P,Q,MN,E12,N1,K1) .
-- for action fkm24
op c-fkm24 :  System Principal Principal Mnonce Nonce Nonce Skey Skey -> Bool
eq c-fkm24(S,P,Q,MN,N1,N2,K1,K2) = N1 \in cnonce(nw(S)) and N2 \in cnonce(nw(S)) and
                                   pr0(K1) = intruder and pr0(K2) = intruder .
eq uv(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = uv(S) .
ceq nw(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = m2(intruder,Q,ca,MN,P,Q,
                                         enc1(K1,MN,N1,P,Q),enc1(K2,MN,N2,P,Q)) , nw(S)
                                      if c-fkm24(S,P,Q,MN,N1,N2,K1,K2) .
eq sk(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = sk(S) .
ceq fkm24(S,P,Q,MN,N1,N2,K1,K2) = S if not c-fkm24(S,P,Q,MN,N1,N2,K1,K2) .
```

図 6.6　侵入者が Message1，Message2 を偽造する定義式

```
-- for action fkm31
op c-fkm31 :  System Principal Principal Mnonce Cipher2 Cipher2 -> Bool
eq c-fkm31(S,P,Q,MN,E21,E22) = E21 \in cci3a(nw(S)) and E22 \in cci3b(nw(S)) .
eq uv(fkm31(S,P,Q,MN,E21,E22)) = uv(S) .
ceq nw(fkm31(S,P,Q,MN,E21,E22)) = m3(intruder,ca,P,MN,E21,E22) , nw(S)
                                  if c-fkm31(S,P,Q,MN,E21,E22) .
eq sk(fkm31(S,P,Q,MN,E21,E22)) = sk(S) .
ceq fkm31(S,P,Q,MN,E21,E22) = S if not c-fkm31(S,P,Q,MN,E21,E22) .

-- for action fkm32
op c-fkm32 :  System Principal Principal Mnonce Cipher2 Nonce Skey Key -> Bool
eq c-fkm32(S,P,Q,MN,E21,N2,K2,K) = E21 \in cci3a(nw(S)) and N2 \in cnonce(nw(S)) and
                                   pr0(K2) = intruder and K \in ckey(nw(S)) .
eq uv(fkm32(S,P,Q,MN,E21,N2,K2,K)) = uv(S) .
ceq nw(fkm32(S,P,Q,MN,E21,N2,K2,K)) = m3(intruder,ca,Q,MN,E21,enc2(K2,N2,K)) , nw(S)
                                      if c-fkm32(S,P,Q,MN,E21,N2,K2,K) .
eq sk(fkm32(S,P,Q,MN,E21,N2,K2,K)) = sk(S) .
ceq fkm32(S,P,Q,MN,E21,N2,K2,K) = S if not c-fkm32(S,P,Q,MN,E21,N2,K2,K) .

-- for action fkm33
op c-fkm33 :  System Principal Principal Mnonce Cipher2 Nonce Skey Key -> Bool
eq c-fkm33(S,P,Q,MN,E22,N1,K1,K) = E22 \in cci3b(nw(S)) and N1 \in cnonce(nw(S)) and
                                   pr0(K1) = intruder and K \in ckey(nw(S)) .
eq uv(fkm33(S,P,Q,MN,E22,N1,K1,K)) = uv(S) .
ceq nw(fkm33(S,P,Q,MN,E22,N1,K1,K)) = m3(intruder,ca,Q,MN,enc2(K1,N1,K),E22) , nw(S)
                                      if c-fkm33(S,P,Q,MN,E22,N1,K1,K) .
eq sk(fkm33(S,P,Q,MN,E22,N1,K1,K)) = sk(S) .
ceq fkm33(S,P,Q,MN,E22,N1,K1,K) = S if not c-fkm33(S,P,Q,MN,E22,N1,K1,K) .

-- for action fkm34
op c-fkm34 :  System Principal Principal Mnonce Nonce Nonce Skey Skey Key -> Bool
eq c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) = N1 \in cnonce(nw(S)) and N2 \in cnonce(nw(S)) and
                                     pr0(K1) = intruder and
                                     pr0(K2) = intruder and K \in ckey(nw(S)) .
eq uv(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = uv(S) .
ceq nw(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = m3(intruder,ca,Q,MN,enc2(K1,N1,K),enc2(K2,N2,K)) , nw(S)
                                        if c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) .
eq sk(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = sk(S) .
ceq fkm34(S,P,Q,MN,N1,N2,K1,K2,K) = S if not c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) .

-- for actin fkm41
op c-fkm41 :  System Principal Principal Mnonce Cipher2 -> Bool
eq c-fkm41(S,P,Q,MN,E21) = E21 \in cci4(nw(S)) .
eq uv(fkm41(S,P,Q,MN,E21)) = uv(S) .
ceq nw(fkm41(S,P,Q,MN,E21)) = m4(intruder,Q,P,MN,E21) ,nw(S)
                        if c-fkm41(S,P,Q,MN,E21) .
eq sk(fkm41(S,P,Q,MN,E21)) = sk(S) .
ceq fkm41(S,P,Q,MN,E21) = S if not c-fkm41(S,P,Q,MN,E21) .

-- for actino fkm42
op c-fkm42 :  System Principal Principal Mnonce Nonce Skey Key -> Bool
eq c-fkm42(S,P,Q,MN,N1,K1,K) = N1 \in cnonce(nw(S)) and pr0(K1) = intruder and K \in ckey(nw(S)) .
eq uv(fkm42(S,P,Q,MN,N1,K1,K)) = uv(S) .
ceq nw(fkm42(S,P,Q,MN,N1,K1,K)) = m4(intruder,Q,P,MN,enc2(K1,N1,K)) , nw(S)
                        if c-fkm42(S,P,Q,MN,N1,K1,K) .
eq sk(fkm42(S,P,Q,MN,N1,K1,K)) = sk(S) .
ceq fkm42(S,P,Q,MN,N1,K1,K) = S if not c-fkm42(S,P,Q,MN,N1,K1,K) .
```

図 6.7　侵入者が Message3，Message4 を偽造する定義式

6.4 モデルの修正・改良点

セキュリティプロトコルの仕様の記述は仕様の検証・修正を繰り返すことによって完成させた．本節では検証作業を進める上で見つかった仕様の不具合などの修正・改良点について記述する．ただし，記述間違い等の単純なミスについては除いている．

6.4.1 2種類のノンスを別のソートで定義

Otway-Rees 認証プロトコルでは 2 種類のノンスを利用する．2 種類のノンスにはともにノンスの生成者，どの主体のためのノンスか，およびノンスたらしめる値が定義してある．2 種類のノンス N と N_A, N_B はともに同じデータ構成をしているが，ノンス N はプロトコルの識別子を表わすもので，プロトコルにおいて，暗号文中と平文であるメッセージ中のどちらにも現われる．一方ノンス N_A, N_B は認証局と認証を行うために利用し，プロトコル中では暗号文中にしか存在しない．すなわち，ノンス N はネットワークを覗けば取得できる情報であるのに対し，ノンス N_A, N_B は暗号文の暗号鍵を持っていなければ取得できない情報である．従って 2 種類のノンスは別のソートとして定義しなければならない．

6.4.2 共通鍵を2種類のソートで定義

プロトコルで使用するのは共通鍵のみであるが，一方は主体と認証局との共通鍵でプロトコル中でメッセージを暗号化するのに利用し，一方は認証局が発行する共通鍵でプロトコルでは暗号文中に含まれているデータとして利用する．前者はプロトコルの仮定においてあらかじめ共有していると定義している．また，メッセージの暗号化に利用するがその鍵自体はネットワーク中に流れることはいっさいない．後者はこのプロトコルの目的となるもので，ネットワークでは暗号文中に含まれている．従って同種の共通鍵であるとしても，その利用法が違うため別のソートとして定義し直した．

6.4.3 侵入者が利用できるノンス N の定義の削除

モジュール NETWORK では侵入者がネットワークから収集する情報について定義してある．ノンス N については平文メッセージ上に現われることから，プロトコルでは誰にでも収集することが出来る．ノンス N は識別子 I_A, I_B と同様の扱いをすることができ，侵入者がネットワークから収集する情報として定義する必要はない．

6.4.4 侵入者が利用できるノンス N_A, N_B の定義変更

モジュール NETWORK で侵入者がネットワークからノンス N_A, N_B を収集する定義において，侵入者がノンスを収集するための条件として，

```
m1?(M) and pr0(skey(ci1(M))) = intruder and nonce(ci1(M)) = N
```

のように定義していた．しかし，暗号文が侵入者と認証局との共通鍵で暗号化されていれば，そのノンスは侵入者自身が作ったものに間違いなくノンスの確認をする必要がなくなる．また，検証時に CafeOBJ 処理系で二重に状態を確認してしまうことになる．従って上記の条件から

```
nonce(ci1(M)) = N
```

の部分は取り除く．

6.4.5 侵入者がメッセージを偽造する遷移規則の変更

侵入者がメッセージを偽造する遷移規則は当初，侵入者が暗号文，ノンス，共通鍵それぞれを入手したときにその事後状態として偽造したメッセージをネットワークに追加するようにしていた．

メッセージを偽造するにはメッセージを構成する要素すべてを入手している必要がある．メッセージの構成要素のうち暗号文以外のデータについては平文で流れているため誰にでも入手することが出来る．これよりメッセージを偽造するには暗号文を入手もしくは偽造する必要がある．暗号文を偽造するためには暗号文の構成要素である情報を入手しなければならない．

したがって，メッセージを偽造する場合は次のように場合分けをした遷移規則を宣言しなければならない．

- 侵入者が暗号文が利用可能である
- 侵入者が暗号文を偽造するのに必要なノンスと共通鍵を利用可能である

メッセージ中に暗号文が 2 つある Message2，Message3 の場合は

- 侵入者が 2 つの暗号文が利用である
- 侵入者が 1 つの暗号文と，もう一つの暗号文を偽造するのに必要なノンス，共通鍵が利用可能である
- 侵入者が 2 つの暗号文を偽造するのに必要なノンス，共通鍵が利用可能である

の 3 つの場合に分けられ，すなわち 4 つの作用演算を用意する必要がある．

6.4.6 ノンス，共通鍵の定義の等式変更

2 種類のノンス，および共通鍵の定義において各データが等しいことを判定するため次のように等式を宣言していた．

```
ノンス  :  eq (nonce(PS1,PD1,V1) = nonce(PS2,PD2,V2))
              = (PS1 = PS2 and PD1 = PD2 and V1 = V2) .
共通鍵  :  eq (key(P1,P2,PK1) = key(P3,P4,PK2))
              = (P1 = P3 and P2 = P4 and PK1 = PK2) .
```

(PS1,PD1,V1,PS2,PD2,V2 は CafeOBJ 変数)

しかし作用演算の効力条件において，ノンスおよび共通鍵に関する条件において不具合を生じた．効力条件は

```
mnonce(M2) = mnonce(ci2a(M2))
K \in ckey(nw(S))
```

などのように宣言している．そのため，各定義の等式での PS1,PD1,V1 などの CafeOBJ 変数がうまく特定できていないと考えられる．そこで，ノンス，共通鍵の各項が特定できるように次のように宣言し直した．

```
ノンス  :  eq (N1 = N2) = (creator(N1) = creator(N2) and
                           reciver(N1) = reciver(N2) and v(N1) = v(N2)) .
共通鍵  :  eq (K1 = K2)
              = (pr1(K1) = pr1(K2) and pr2(K1) = pr2(K2) and pk(K1) = pk(K2)) .
```

(N1,N2,K1,K2 は CafeOBJ 変数)

第7章　Otway-Rees認証プロトコルの検証

7.1　検証する性質

Otway-Rees認証プロトコルでは，次の表明が成り立つことを記述したCafeOBJ文書を基に，証明譜を記述し，CafeOBJ処理系で実行することで検証する．

表明 侵入者は，侵入者とは異なる主体と侵入者とは異なる別の主体のために認証局が発行した共通鍵を取得することはない．

このことを検証することで，認証局によって発行された共通鍵 K_{AB} は，認証局，主体 A，B の三者の間だけで共有されることを保証する．これにより，主体 A，B は共通鍵 K_{AB} を利用して秘密の通信ができるようになる．

この表明は，侵入者が収集する共通鍵の集まりに K_{AB} のような共通鍵が含まれることはない，すなわち侵入者が収集する共通鍵の集まりに含まれる共通鍵は，侵入者と侵入者とは異なる別の主体のために認証局が発行した共通鍵のみである，といえる．

表明は次のように項で表わすことができる．

表明 1．任意の到達可能な `s : Sys` および任意の `k : Key` にたいして，

```
k \in ckey(nw(s)) implies (pr1(k) = intruder or pr2(k) = intruder)
```

_implies_は論理含意を表わす演算子，_or_は宣言を表わす演算子．

前章のモデル化より，Otway-Rees認証プロトコルの状態 `s` が与えられると，侵入者が収集する共通鍵は項 `ckey(nw(S))` で表わされる．共通鍵 `k` が与えられると，その共通鍵が侵入者のために発行されたものであれば `pr1(k)` と `pr2(k)` のどちらかが `intruder` と等しい．

7.2　補題

表明1の検証は単独でこの帰納法により検証を試みようとすると，帰納法の仮定が弱いため，帰納段階での証明ができなくなってしまう．そのため，次のような補題が必要となる．

補題 1. 任意の到達可能な s : Sys および任意の e31 : Cipher2 に対して,

```
e31 \in cci3a(nw(s)) implies not(pr0(skey(e31)) = intruder)
```

補題 2. 任意の到達可能な s : Sys および任意の e32 : Cipher2 に対して,

```
e32 \in cci3b(nw(s)) implies not(pr0(skey(e32)) = intruder)
```

補題 3. 任意の到達可能な s : Sys および任意の e4 : Cipher2 に対して,

```
e4 \in cci4(nw(s)) implies not(pr0(skey(e4)) = intruder)
```

補題 1 から 3 は侵入者が収集する暗号文は侵入者と認証局との共通鍵で暗号化されていない暗号文である，ことを意味している．暗号文の暗号化鍵が認証局と侵入者との共通鍵で暗号化されている場合は，侵入者は復号することが可能であるからである．

7.3 表明の検証

4.4 節で記述した検証法に基づき検証を行った．表明の検証を行うためにまずモジュール INV を宣言する．モジュール INV には前述の表明 1 の述語を表わす演算子 inv100 を宣言する．同時にそれらを定義する等式を宣言する．

```
op inv040 : System Cipher2 -> Bool
op inv050 : System Cipher2 -> Bool
op inv060 : System Cipher2 -> Bool
op inv100 : System Key -> Bool
eq inv040(S,E31) = (E31 \in cci3a(nw(S)) implies not(pr0(skey(E31)) = intruder)) .
eq inv050(S,E32) = (E32 \in cci3b(nw(S)) implies not(pr0(skey(E32)) = intruder)) .
eq inv060(S,E4) = (E4 \in cci4(nw(S)) implies not(pr0(skey(E4)) = intruder)) .
eq inv100(S,K) = (K \in ckey(nw(S)) implies (pr1(K) = intruder or pr2(K) = intruder)) .
```

ここで S はソート System の CafeOBJ 変数であり，K はソート Key の CafeOBJ 変数である．

次にモジュール ISTEP で帰納段階で示す述語を表わす演算子 istep100 を宣言し，同時にそれらを定義する等式を宣言する．

```
op istep040 : Cipher2 -> Bool
op istep050 : Cipher2 -> Bool
op istep060 : Cipher2 -> Bool
op istep100 : Key -> Bool
eq istep040(E31) = inv040(s,E31) implies inv040(s',E31) .
eq istep050(E32) = inv050(s,E32) implies inv050(s',E32) .
eq istep060(E4) = inv060(s,E4) implies inv060(s',E4) .
eq istep100(K) = inv100(s,K) implies inv100(s',K) .
```

ここで，s は任意の状態を，s' は s で表わされる状態から遷移規則が適応された事後状態を表わす．定数 s,s' は次のように宣言する．

```
ops s s' : -> Bool
```

次にすべての遷移規則に対して帰納段階の証明を行う．本稿では表明 1 の証明譜について記述する．まず，任意の初期状態において inv100 で表わされる述語が成り立つことを示す．証明譜の記述は次のように行う．

```
open INV
  red inv100(init,k) .
close
```

帰納段階では各作用演算で表わされる遷移規則が表明の性質を保持することを示すとき状態空間を複数に分割する．表 7.1〜表 7.8 は各作用演算を場合分けをした結果である．なお，作用演算 mes1，mes2，fkm11，fkm12，fkm21，fkm22，fkm23，fkm34 については Message1 及び Message2 に関する作用演算であるが，表明の共通鍵が存在しないため遷移規則の効力条件が真である場合と偽である場合の 2 通りのみであった．

表 7.1: 作用演算 mes3 で表わされる遷移規則の帰納段階における場合分け

1	c-med3(s,mes10,pk10))	not(k = key(id1(mes10),id2(mes10),pk10))		
2		k = key(id1(mes10),id2(mes10),pk10	id1(mes10) = intruder)	
3			not(id1(mes10) = intruder))	id2(mes10) = intruder
4				not(id2(mes10) = intruder)
5	not c-mes3(s,mes10,pk10))			

表 7.2: 作用演算 mes4 で表わされる遷移規則の帰納段階における場合分け

1	c-mes4(s,q10,mes10,mes20,mes30)	not(k = key(ci3a(mes30)))	
2		k = key(ci3a(mes30))	pr0(skey(ci3a(mes30))) = intruder
3			not(pr0(skey(ci3a(mes30))) = intruder)
4	not c-mes4(sq10,mes10,mes20,mes30)		

表 7.3: 作用演算 fkm31 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm31(s,p10,q10,m10,e10,e20))	pr0(skey(e10)) = intruder	
2		not(pr0(skey(e10)) = intruder)	pr0(skey(e20)) = intruder
3			not(pr0(skey(e20)) = intruder)
4	not c-fkm31(s,p10,q10,m10,e10,e20))		

表 7.4: 作用演算 fkm32 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm32(s,p10,q10,m10,e10,n20,k20,k30))	k = k30	
2		not(k = k30)	not(pr0(skey(e10)) = intruder)
3			pr0(skey(e20)) = intruder
4	not c-fkm3(s,p10,q10,m10,e10,n20,k20,k30))		

表 7.5: 作用演算 fkm33 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm33(s,p10,q10,m10,e20,n10,k10,k30))	k = k30	
2		not(k = k30)	not(pr0(skey(e20)) = intruder)
3			pr0(skey(e20)) = intruder
4	not c-fkm33(s,p10,q10,m10,e20,n10,k20,k10))		

表 7.6: 作用演算 fkm34 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30))	k = k30
2		not(k = k30)
3	not c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30))	

表 7.7: 作用演算 fkm41 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm41(s,p10,q10,m10,e10))	not(pr0(skey(e10)) = intruder)
2		pr0(skey(e10)) = intruder
3	not c-fkm41(s,p10,q10,m10,e10))	

表 7.8: 作用演算 fkm42 で表わされる遷移規則の帰納段階における場合分け

1	c-fkm42(s,p10,q10,m10,n10,k10,k30))	k = k30
2		not(k = k30)
3	not c-fkm42(s,p10,q10,m10,n10,k10,k30))	

7.3.1 作用演算 fkm33

本稿では表明 1 の作用演算 fkm33 に対応する遷移規則の帰納段階の証明譜について記述する.

任意の到達可能な状態を表わす定数 s，任意の主体を表わす定数 p10，q10，ノンス N を表わす定数 m10．メッセージ中の暗号文を表わす定数 e20，暗号文中のノンスを表わす定数 n10，認証局との共通鍵を表わす定数 k10, 認証局が発行する共通鍵を表わす定数 k30 に対して遷移規則は項 fkm33(s,p10,q10,m10,e20,n10,k20,k30) で表わされる.

遷移規則の効力条件，事後状態を考慮しながら表 7.5 のように状態空間を分割した．場合分けは表 7.5 の上から下，および左から右に向かって証明譜を書きながら行った．各場合は表の行に現われる述語を and で連接した述語で表わされる．例えば，場合 2 は

```
c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
  and not(k = k30) and not(pr0(skey(e20)) = intruder)
```

である任意の状態のことを意味している..

次に各状態の証明譜について示す．各証明譜では，項で表わされる定数を宣言する．続いてに各場合を表わす等式を宣言する．次に等式で s' が状態 s の事後状態でを表わすことを宣言する．最後に red コマンドで証明すべき述語を簡約することで証明の正しさを確認する．ここで true が返ってきたとき，この状態は証明すべき性質を有していることがわかる．

場合 1

場合 1 では議論中の共通鍵を表わす定数 k を定数 k30 に書き換えている．この簡約の結果 true が返ってきたのでこの場合 (空間) では表明 1 の性質を有していることが確認できる．

```
open ISTEP
-- arbitrary objects
  ops p10 q10 : -> Principal .
  op m10 : -> Mnonce .
  op e20 : -> Cipher2 .
  op n10 : -> Nonce .
  op k10 : -> Skey .
  op k30 : -> Key .
-- assumptions
  -- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
  eq e20 \in cci3b(nw(s)) = true .
  eq n10 \in cnonce(nw(s)) = true .
  eq pr0(k10) = intruder .
  eq k30 \in ckey(nw(s)) = true .
  --
  eq k = k30 .
-- successor state
  eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
  red inv050(s,e20) implies istep100(k) .
close
```

場合 2

場合 2 では議論中の共通鍵を表わす定数 k が定数 k30 で表わされる共通鍵と異なることに加えて，e20 で表わされる暗号文が侵入者と認証局との共通鍵で暗号化されていないことを表わしている．簡約の結果この場合においても true が得られた．

```
open ISTEP
-- arbitrary objects
  ops p10 q10 : -> Principal .
  op m10 : -> Mnonce .
  op e20 : -> Cipher2 .
  op n10 : -> Nonce .
  op k10 : -> Skey .
  op k30 : -> Key .
-- assumptions
  -- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
  eq e20 \in cci3b(nw(s)) = true .
  eq n10 \in cnonce(nw(s)) = true .
  eq pr0(k10) = intruder .
```

```
  eq k30 \in ckey(nw(s)) = true .
  --
  eq (k = k30) = false .
  eq not(pr0(skey(e20)) = intruder) .
-- successor state
  eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
  red inv050(s,e20) implies istep100(k) .
close
```

場合 3

場合 3 では e20 で表わされる暗号文が侵入者と認証局との暗号文で暗号化されている．しかし，この場合を簡約しても帰納法の仮定が弱いため，期待通りに true は得られないため，補題が必要となった．必要となった補題は,「収集する暗号文に侵入者の共通鍵で暗号化された暗号文はない」という事実である．補題を用いた簡約の結果 true が得られた．

```
open ISTEP
-- arbitrary objects
  ops p10 q10 : -> Principal .
  op m10 : -> Mnonce .
  op e20 : -> Cipher2 .
  op n10 : -> Nonce .
  op k10 : -> Skey .
  op k30 : -> Key .
-- assumptions
  -- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
  eq e20 \in cci3b(nw(s)) = true .
  eq n10 \in cnonce(nw(s)) = true .
  eq pr0(k10) = intruder .
  eq k30 \in ckey(nw(s)) = true .
  --
  eq (k = k30) = false .
  eq pr0(skey(e20)) = intruder .
-- successor state
  eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
  red inv050(s,e20) implies istep100(k) .
close
```

場合 4

場合 4 は遷移規則の効力条件が偽の場合なので，状態が変化しないことから，表明の性質が保存されることは明らかである．

```
open ISTEP
-- arbitrary objects
  ops p10 q10 : -> Principal .
  op m10 : -> Mnonce .
  op e20 : -> Cipher2 .
  op n10 : -> Nonce .
  op k10 : -> Skey .
  op k30 : -> Key .
-- assumptions
  eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
-- successor state
```

```
  eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
  red istep100(k) .
close
```

遷移規則 fkm33 以外の遷移規則の証明譜についても同様に記述することができ，すべての場合において期待通りに true を得ることができた．また，表明 1 のみだけでなく，補題 1，補題 2，補題 3 についても同様の記述方法により場合分けを行い，証明譜を記述した．その結果，モデルが各補題の性質を有することを確認した．

これらより，本研究でモデル化を行った Otway-Rees 認証プロトコルは表明 1 の性質を有することを確認することができた．すなわち，このモデルにおいて侵入者が入手できる共通鍵は認証局が侵入者と侵入者と異なる別の主体のために発行した共通鍵のみであることがわかる．言い換えると，侵入者が，侵入者とは異なる主体と侵入者とは異なる別の主体のための共通鍵を取得することはないことを検証することができたことになる．

第8章　関連研究

8.1　Otway-Rees認証プロトコルへの攻撃

Otway-Rees認証プロトコルへの攻撃としてメッセージビット数を利用したものがある[1]．Otway-Rees認証プロトコルのメッセージは主体の識別子，ノンス，共通鍵，暗号文などの情報のシーケンスである．メッセージを型情報のないビット列として送信する場合プロトコルに欠陥が発生する場合がある．例えば：

4つのデータ構成子を持つメッセージ
　　Ａ：４ｂｉｔ
　　Ｂ：６ｂｉｔ
　　Ｃ：８ｂｉｔ
　　Ｄ：６ｂｉｔ
があるとする

メッセージとしてビット列
　　０１１００１０００１０１０１０００１１１０１

を受信者が受け取ると
受信者は各データを
　　Ａ：０１１０
　　Ｂ：００１０００
　　Ｃ：１０１０１０００
　　Ｄ：０１１１０１
と認識して受け取る

次のOtway-Rees認証プロトコルで認証局は2つの主体の認証を行い共通鍵を発行し，鍵の分配を行う．しかし，実体の認証もしくは，鍵の確認は行わないとする．すなわちメッセージのビット列が与えられた場合，プロトコルに則ったビット数をそれぞれの情報であると認識する．また，主体と認証局との共通鍵はあらかじめ共有しているものとする．ノンスNはプロトコルの識別子である．

$$
\begin{aligned}
Message1.\quad & A \rightarrow B \quad : N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}} \\
Message2.\quad & B \rightarrow S \quad : N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}}, \{N_B, N, I_A, I_B\}_{K_{BS}} \\
Message3.\quad & S \rightarrow B \quad : N, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
Message4.\quad & B \rightarrow A \quad : N, \{N_A, K_{AB}\}_{K_{AS}}
\end{aligned}
$$

$Otway - Rees$ 認証プロトコル

ここでノンス N のビット数を 32bit, 主体 A，主体 B の各識別子のビット数を 16bit, 認証局が発行する共通鍵 K_{AB} のビット数を 64bit とする．また，暗号文中のデータの順番は上記のプロトコルどおりとする．

こうすることにより次のように不正を行うことが可能となる．

CASE1. 主体 B が不正を行う場合

不正な主体 B を $M(B)$ とする.

$$
\begin{array}{lll}
\text{M1.} \quad A \to M(B) & : & N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}} \\
\text{M2.} \quad M(B) \to S & : & N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}}, \{N_B, N, I_A, I_B\}_{K_{BS}} \\
\text{M3.} \quad S \to M(B) & : & N, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
\text{M4.} \quad M(B) \to A & : & N, \{N_A, N, I_A, I_B\}_{K_{AS}} \\
\text{M5.} \quad A \leftrightarrow B & : & \{anything\ recognizable\}_{\{N, I_A, I_B\}}
\end{array}
$$

Message1 を受け取った主体 B は Message4 で認証局が発行した共通鍵が含まれている暗号文を送らずに，Message1 の暗号文をそのまま送り返す.（Message2，Message3 のステップは省略可）不正な Message4 を受け取った主体 A は，ノンス N，識別子 A, B のビット数の合計 (64bit) が共通鍵のビット数 (64bit) と同じため，暗号文を復号したときに得られるビット列は，期待通りノンス N_A と，認証局が発行した共通鍵が送られてきているものと認識し、ノンス N と識別子 A, B の組み合わせを共通鍵として秘密通信を行ってしまう可能性がある．送られてきた Message4 が Message1 と同一かどうかを確認すれば，正しく認証局に発行してもらった共通鍵かどうかを判断することができるが，確認しなかった場合は，誰にでも入手できるデータを使って暗号通信を行ってしまうことになる.

CASE2. 認証局が不正を行う場合

不正な認証局を $M(S)$ とする.

$$
\begin{array}{lll}
\text{M1.} \quad A \to B & : & N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}} \\
\text{M2.} \quad B \to M(S) & : & N, I_A, I_B, \{N_A, N, I_A, I_B\}_{K_{AS}}, \{N_B, N, I_A, I_B\}_{K_{BS}} \\
\text{M3.} \quad M(S) \to B & : & N, \{N_A, N, I_A, I_B\}_{K_{AS}}, \{N_B, N, I_A, I_B\}_{K_{BS}} \\
\text{M4.} \quad B \to A & : & N, \{N_A, N, I_A, I_B\}_{K_{AS}} \\
\text{M5.} \quad A \leftrightarrow B & : & \{anything\ recognizable\}_{\{N, I_A, I_B\}}
\end{array}
$$

Message2 を受け取った不正な認証局は共通鍵を発行せず，主体 A，B が作ったそれぞれの暗号文をそのまま Message3 の暗号文として主体 B に送り返す．Message3，Message4 をそれぞれ受け取った 2 つの主体は CASE1 の時と同様にノンス N と識別子 A, B の組み合わせが共通鍵と認識し，以後の秘密通信に利用する可能性がある.

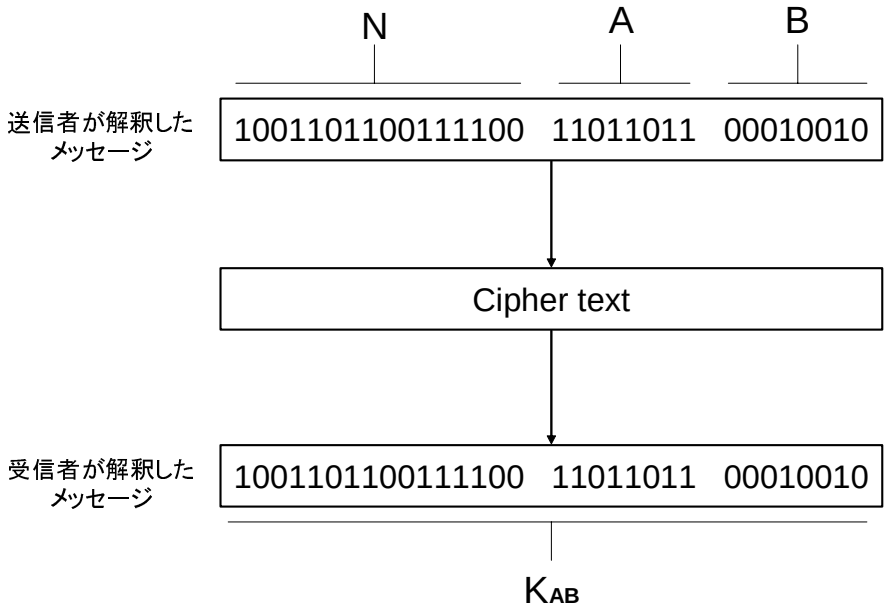

図 8.1: ビット数による不正を行う仕組み

以上の不正行為は主体 B および認証局が不正を行うだけでなく，侵入者がネットワークを盗聴することで行うこともできる．これらの欠陥は各データを単にビット数をカウントするだけで認識することによって引き起こる可能性がある．本研究では主体の識別子，各ノンス，共通鍵，および暗号鍵は個々の項 (term) として定義しているので，ビットカウントによる不正をはたらいた場合，各項をチェックする段階で不具合[1]が起こったことがわかりプロトコルの安全性には影響しない．

[1] ここでいう不具合とはプロトコルの欠陥ではなく，通信路上でのエラーや，侵入者による攻撃を検出することである．

8.2 NSLPK 認証プロトコル

セキュリティプロトコルを形式的に検証する手法は 1995 年に Lowe により NSPK プロトコルの欠陥が発見されたことによってその有効性に期待が持たれている．

観測遷移機械を用いた NSLPK 認証プロトコルの安全性の検証はすでに行われており [7]，ノンスの秘密性，通信の信頼性が確認されており，安全性を満たすことを検証されている．

本研究で例題として取り上げた Otway-Rees 認証プロトコルと NSLPK 認証プロトコルとでは，どちらも認証を行い，秘密通信を行うための鍵を共有するという目的は同じである．しかし，Otway-Rees 認証プロトコルでは認証局が存在[2]しており，認証の仕方，鍵の共有の仕方に大きな違いがある．2 つのプロトコルの主な違いは次のようになっている．

- 暗号化鍵

  Otway-Rees 認証プロトコルでは双方向性の鍵，共通鍵暗号を利用している．したがって鍵の共有者以外に鍵を知られてはいけない．NSLPK 認証プロトコルは一方向性関数を利用した公開鍵暗号を利用している．公開鍵は広く一般に公開されており，誰にでもメッセージを暗号化できる．復号に関しては主体が他の主体に知られないように管理しなければならない秘密鍵を用いる．

- 認証方法

  Otway-Rees 認証プロトコルは通信を行う主体同士が直接認証を行うのではなく，認証局を介在して行う．ここで，認証局は信頼できるものでないと正しい認証が行われない可能性がある．各主体は自分と認証局との共通鍵でノンスを暗号化する．認証局からの暗号文を受け取ったら，復号をして暗号文中のノンスが一致することを確認する．暗号文は共通鍵で暗合されているので自分の生成したノンスは認証局にしか知り得ない．これにより認証局と認証を行う．

  NSLPK 認証プロトコルは生成したノンスを通信相手の公開鍵で暗号化し送信する．通信相手から自分の公開鍵で暗号化された暗号文を受け取ったら自分の秘密鍵で復号し，暗号文中に自分の生成したノンスがあることを確認することで認証を行う．秘密鍵は他人に知られないように管理されているので暗号文を第三者によって復号されることはない．

- 鍵の共有方法

  Otway-Rees 認証プロトコルでは認証局が主体と認証を行うためのノンスを暗号化する際に，鍵を発行して暗号文中に鍵を入れて送信する．NSLPK 認証プロトコルではプロトコル終了後，二つの主体の間で共有しており，その二つのノンスを基に各主体が共通鍵を生成する．

[2]正確には NSLPK 認証プロトコルにも認証局が存在している．認証局は各主体の公開鍵を管理しており，その配布を行う．ここでは，互いの公開鍵はすでに保有していることを前提としている．

- ノンス

  Otway-Rees 認証プロトコルではプロトコルの識別子として一般に公開されるノンスと，認証に利用する第三者に知られてはならないノンスの 2 種類のノンスを利用する．NSLPK 認証プロトコルは認証に利用する第三者に知られてはいけないノンスの 1 種類を利用する．

- メッセージ

  NSLPK 認証プロトコルでは送信する情報はすべて暗号化されている．Otway-Rees 認証プロトコルでは各主体の識別子とプロトコルの識別子であるノンスは暗号文中と平文中に現われる．

Otway-Rees 認証プロトコルのモデル化，及び検証は NSLPK 認証プロトコルと比べて主体，認証局，侵入者，メッセージ，ネットワーク等のデータ型の宣言，プロトコルのモデル化はプロトコルの差異はあるものの同様の手段により形式化，及び検証することができると考えられる．

第9章　まとめと今後の課題

9.1　まとめ

本研究では，Otway-Rees 認証プロトコルを例題にセキュリティプロトコルを観測遷移機械でモデル化を行い，CafeOBJ でモデルを記述した．次に検証したい性質を CafeOBJ で表現し，その性質をプロトコルが有していることを示す証明譜を CafeOBJ で記述した．続いて CafeOBJ 処理系で証明譜を実行することで証明の正しさの確認を行った．

観測遷移機械と CafeOBJ の組み合わせによる検証はモデルの記述，および証明譜の記述を検証者自身が行うことにより，プロトコルの理解が容易となるほか，使用するデータ型を適切に定義することができ，記述したモデルの仕様の可読性に優れている．

観測遷移機械でのモデル化はまず主体や侵入者の存在などのように対象となるシステムの前提条件を仮定する．次にプロトコルで使用するデータ型の定義を行う．続いて観察可能なプロトコル内部の値を決めプロトコルの振舞を遷移規則としてモデル化した．

検証作業はまず，検証したい性質を CafeOBJ の項として記述をする．次に CafeOBJ で証明譜を記述し，CafeOBJ 処理系で証明譜を実行させることで検証を行った．

本研究で検証した性質は Otway-Rees 認証プロトコルの秘匿性についてである．すなわち，Otway-Rees 認証プロトコルで認証局が発行する共通鍵を不正な侵入者が取得することはないということである．その結果，Otway-Rees 認証プロトコルが共通鍵の秘匿性についての性質を有していることを確認することことができた．

仕様の検証はその作業工程においては，記述ミスや，モデル自体の不具合が発見されるなどがあり，仕様の修正と検証を繰り返し行った．検証作業における場合分け・補題発見の作業は遷移規則の効力条件，事後状態に基づいてある程度までは規則的に行うことができるが，検証を行う検証者自身が行うことから，検証者の経験に依存してしまうためもっとも苦労した部分であった．

9.2　今後の課題

本研究の今後の課題としては次のようなことが考えられる．

- Otway-Rees 認証プロトコルの信頼性に関しての検証
  - 本研究では発行された共通鍵が不正に侵入者によって取得されることはないというプロトコルの秘匿性についての検証を行った．しかし，プロトコルが完

全に安全性を有していることを保証するためには，秘匿性の保証だけでなく，メッセージがプロトコルに則って正しく送信されることを確認する，すなわちプロトコルが信頼性を有していることの保証もしなければならない．

- 現実的なプロトコルへの適用
  - 本研究で例題として取り上げた Otway-Rees 認証プロトコルおよび観測遷移機械を用いてすでに検証が行われている NSLPK 認証プロトコルの例を比較し，認証プロトコル一般に対する形式的記述・検証の方法論を確立させ，SSL，SSH などの現実的に利用されているようなプロトコルに適用させることも今後の課題の一つである．

謝辞

本研究を進めるにあたりご指導いただいた二木厚吉教授に深く感謝いたします．緒方和博客員研究員，中村正樹助手には本研究についての有益な御助言をしていただきました．心より感謝いたします．また，ゼミを通して研究に関する議論につきあっていただいた言語設計学講座の皆様に感謝いたします．

参考文献

[1] David Basin, Security Protocols I, Department of Computer Science, ETH Zurich `http://www.infsec.ethz.ch/education/ws0304/infsec_material/protocols1.pdf`, 2003

[2] Kazuhiro Ogata, Kokichi Futatsugi, Rewriting-Based verification of authentication protocols. *the 4th International Workshop in Rewriting Logic and its Applications(WRLA 2002))*, Electronic Notes in theoretical Computer Science 71, 15 pages. Elsevier Science Publishers, 2002.

[3] Lowe,G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR, *3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS,Vol.1055, Springer, 1996, pp.147-166.

[4] Paulson,L. C. The Inductive Approach to Verifying Cryptographic Protocols, *Journal of Computer Security*, Vol.6, pp.85-128, 1998.

[5] M.Burrows, M.Abadi, and R.M.Needham, A logic of authentication, *Proceedings of the Royal Society of London*, 426:pp.233-271, 1989.

[6] Nakagawa,A., T.Sawada and K.Futatsugi, CafeOBJ User's Manual — Ver. 1.4 —, `http://www.ldl.jaist.ac.jp/cafeobj/doc`, 1999.

[7] 緒方和博, 二木厚吉, 書き換えによるセキュリティプロトコルの帰納的検証, コンピュータソフトウェア,vol.20, No3, pp.54-72, 2003.

[8] 金城直樹セキュリティプロトコルの代数モデルに基づく形式化, 北陸先端科学技術大学院大学, 修士論文, 2001.

[9] セキュリティ研究会まるごと図解　最新インターネットセキュリティがわかる 技術評論社, 2000.

付 録A　Otway-Reesプロトコルの仕様

```
------------------------------------------------------------------------------
--
-- Otway-Rees Shared-key Authentication Protocol
--
-- Msg1 A --> B : M,A,B,{N_A,M,A,B}k_A
-- Msg2 B --> S : M,A,B,{N_A,M,A,B}k_A,{N_B,M,A,B}k_B
-- Msg3 S --> B : M,{N_A,k_AB}K_A,{N_b,k_AB}k_B
-- Msg4 B --> A : M,{N_A,K_AB}
--
------------------------------------------------------------------------------

--
-- 主体の定義
-- 侵入者は汎用の侵入者としてモデル化する
--
mod* PRINCIPAL principal-sort Principal {
[Principal]
op intruder : -> Principal
op _=_ : Principal Principal -> Bool {comm}
var P : Principal
eq (P = P) = true .
}


--
-- 認証局の定義
-- 侵入者は汎用の侵入者としてモデル化する
--
mod* SERVER principal-sort Server {
[Server]
op ca : -> Server
op intruder : -> Server
op _=_ : Server Server -> Bool {comm}
var S : Server
eq (S = S) = true .
}


--
-- MNonce の値（類推不可能）の定義
-- (1 つのセッションの確認のために使用)
--
mod* MVALUE principal-sort Mvalue {
[Mvalue]
op _=_ : Mvalue Mvalue -> Bool {comm}
var MV : Mvalue
eq (MV = MV) = true .
}


--
-- Mnonce の定義（一つのセッションを保障するために使用する）
-- mnonce(p,q,nv) は主体 p が主体 q のために生成したノンスで、nv がそのノンスの唯一性を表す
```

```
--
mod! MNONCE principal-sort Mnonce {
pr(PRINCIPAL + MVALUE)
[Mnonce]
op mnonce : Principal Principal Mvalue -> Mnonce
op creator : Mnonce -> Principal
op reciver : Mnonce -> Principal
op mv : Mnonce -> Mvalue
op _=_ : Mnonce Mnonce -> Bool {comm}
vars PS1 PD1 : Principal
vars V1 : Mvalue
vars MN1 MN2 : Mnonce
eq creator(mnonce(PS1,PD1,V1)) = PS1 .
eq reciver(mnonce(PS1,PD1,V1)) = PD1 .
eq mv(mnonce(PS1,PD1,V1)) = V1 .
eq (MN1 = MN2) = (creator(MN1) = creator(MN2) and reciver(MN1) = reciver(MN2) and mv(MN1) = mv(MN2)) .
}


--
-- Nonce の値（類推不可能）の定義
--（認証局との認証に使用）
--
mod* NVALUE principal-sort Nvalue {
[Nvalue]
op _=_ : Nvalue Nvalue -> Bool {comm}
var NV : Nvalue
eq (NV = NV) = true .
}


--
-- Nonce の定義（主体が認証局との認証のために使用する）
-- nonce(p,q,nv)は主体 p が主体 q のために生成したノンスで、nv がそのノンスの唯一性を表す
--
mod! NONCE principal-sort Nonce {
pr(PRINCIPAL + NVALUE)
[Nonce]
op nonce : Principal Principal Nvalue -> Nonce
op creator : Nonce -> Principal
op reciver : Nonce -> Principal
op v : Nonce -> Nvalue
op _=_ : Nonce Nonce -> Bool {comm}
vars PS1 PD1 : Principal
var V1 : Nvalue
vars N1 N2 : Nonce
eq creator(nonce(PS1,PD1,V1)) = PS1 .
eq reciver(nonce(PS1,PD1,V1)) = PD1 .
eq v(nonce(PS1,PD1,V1)) = V1 .
eq (N1 = N2) = (creator(N1) = creator(N2) and reciver(N1) = reciver(N2) and v(N1) = v(N2)) .
}


--
-- KEY のセッション毎の唯一性を与えるための定義
--
mod* PUBKEY principal-sort Pubkey {
[Pubkey]
op _=_ : Pubkey Pubkey -> Bool {comm}
var PK : Pubkey
eq (PK = PK) = true .
}


--
-- 認証局が生成する２つの主体のための共通鍵の定義
```

```
-- key(p,q,pk) は認証局が発行した主体 p と主体 q の共通鍵であり
-- pk はそのセッションだけの鍵であることを現す
--
mod! KEY principal-sort Key {
pr(PRINCIPAL + PUBKEY)
[Key]
op key : Principal Principal Pubkey -> Key
ops pr1 pr2 : Key -> Principal
op pk : Key -> Pubkey
op _=_ : Key Key -> Bool {comm}
vars K1 K2 : Key
vars P1 P2 : Principal
vars PK1 : Pubkey
eq pr1(key(P1,P2,PK1)) = P1 .
eq pr2(key(P1,P2,PK1)) = P2 .
eq pk(key(P1,P2,PK1)) = PK1 .
eq (K1 = K2) = (pr1(K1) = pr1(K2) and pr2(K1) = pr2(K2) and pk(K1) = pk(K2)) .
}


--
-- 認証局と主体との共通鍵
--
mod! SKEY principal-sort Skey {
pr(PRINCIPAL)
[Skey]
op skey : Principal -> Skey
op pr0 : Skey -> Principal
op _=_ : Skey Skey -> Bool {comm}
vars K1 K2 : Skey
var P1 : Principal
eq pr0(skey(P1)) = P1 .
eq (K1 = K2) = (pr0(K1) = pr0(K2)) .
}


--
-- Message1,Message2 に現れる暗号文
-- enc1(sk(p),m,n,p,q) は、主体 p と認証局との共通鍵 sk(p) で暗号化された暗号文で
-- m はノンス M、n はノンス N、p,q は主体をそれぞれ表している
--
mod! CIPHER1 principal-sort Cipher1 {
pr(PRINCIPAL + NONCE + MNONCE + KEY + SKEY)
[Cipher1]
op enc1 : Skey Mnonce Nonce Principal Principal -> Cipher1
op skey : Cipher1 -> Skey
op mnonce : Cipher1 -> Mnonce
op nonce : Cipher1 -> Nonce
ops p q   : Cipher1 -> Principal
op _=_ : Cipher1 Cipher1 -> Bool {comm}
vars SK1 SK2 : Skey
vars M1 M2 : Mnonce
vars N1 N2 : Nonce
vars PS1 PS2 PR1 PR2 : Principal
eq skey(enc1(SK1,M1,N1,PS1,PR1)) = SK1 .
eq mnonce(enc1(SK1,M1,N1,PS1,PR1)) = M1 .
eq nonce(enc1(SK1,M1,N1,PS1,PR1)) = N1 .
eq p(enc1(SK1,M1,N1,PS1,PR1)) = PS1 .
eq q(enc1(SK1,M1,N1,PS1,PR1)) = PR1 .
eq (enc1(SK1,M1,N1,PS1,PR1) = enc1(SK2,M2,N2,PS2,PR2))
= (SK1 = SK2 and M1 = M2 and N1 = N2 and PS1 = PS2 and PR1 = PR2) .
}


--
-- Message3,Message4 に現れる暗号文
```

```
-- enc2(sk(p),n,k(p q)) は、主体 p と認証局との共通鍵 sk(p) で暗号化された暗号文で
-- n はノンス N を、k(p q) は認証局が発行した主体 p と主体 q との共通鍵を表している
--
mod! CIPHER2 principal-sort Cipher2 {
pr(NONCE + PRINCIPAL + KEY + SKEY)
[Cipher2]
op enc2 : Skey Nonce Key -> Cipher2
op skey : Cipher2 -> Skey
op key : Cipher2 -> Key
op nonce : Cipher2 -> Nonce
op _=_ : Cipher2 Cipher2 -> Bool {comm}
vars SK1 SK2 : Skey
vars K1 K2 : Key
vars N1 N2 : Nonce
eq skey(enc2(SK1,N1,K1)) = SK1 .
eq key(enc2(SK1,N1,K1)) = K1 .
eq nonce(enc2(SK1,N1,K1)) = N1 .
eq (enc2(SK1,N1,K1) = enc2(SK2,N2,K2))
 = (SK1 = SK2 and N1 = N2 and K1 = K2) .
}


--
-- メッセージのための可視ソート
-- m*(creator,ssender,reciver,send-messagae)
--
mod! MSG principal-sort Msg{
pr(PRINCIPAL + SERVER + CIPHER1 + CIPHER2 + MNONCE)
[Msg]
--    creator   sender    receiver  send-messeage
-- ------------------------------------------------------------------------
op m1 : Principal Principal Principal Mnonce Principal Principal Cipher1 -> Msg
op m2 : Principal Principal Server    Mnonce Principal Principal Cipher1 Cipher1 -> Msg
op m3 : Server    Server    Principal Mnonce Cipher2   Cipher2 -> Msg
op m4 : Principal Principal Principal Mnonce Cipher2 -> Msg
--
ops m1? m2? m3? m4? : Msg -> Bool
ops creator1 sender1 reciver1 id1 id2 : Msg -> Principal
ops creator2 sender2 reciver2 : Msg -> Server
op mnonce : Msg -> Mnonce
ops ci1 ci2a ci2b : Msg -> Cipher1
ops ci3a ci3b ci4 : Msg -> Cipher2
op _=_ : Msg Msg -> Bool {comm}
--
vars M M1 M2 : Msg
vars CP SP RP P1 P2 : Principal
vars CS SS RS : Server
var MN : Mnonce
vars E1 E21 E22 : Cipher1
vars E31 E32 E4 : Cipher2
--
-- 与えられたメッセージがどの種のメッセージかを判定する
eq m1?(m1(CP,SP,RP,MN,P1,P2,E1)) = true .
eq m1?(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = false .
eq m1?(m3(CS,SS,RP,MN,E31,E32)) = false .
eq m1?(m4(CP,SP,RP,MN,E4)) = false .
--
eq m2?(m1(CP,SP,RP,MN,P1,P2,E1)) = false .
eq m2?(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = true .
eq m2?(m3(CS,SS,RP,MN,E31,E32)) = false .
eq m2?(m4(CP,SP,RP,MN,E4)) = false .
--
eq m3?(m1(CP,SP,RP,MN,P1,P2,E1)) = false .
eq m3?(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = false .
eq m3?(m3(CS,SS,RP,MN,E31,E32)) = true .
eq m3?(m4(CP,SP,RP,MN,E4)) = false .
```

```
--
eq m4?(m1(CP,SP,RP,MN,P1,P2,E1)) = false .
eq m4?(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = false .
eq m4?(m3(CS,SS,RP,MN,E31,E32)) = false .
eq m4?(m4(CP,SP,RP,MN,E4)) = true .
-- メッセージを送信 (作成) した主体
eq creator1(m1(CP,SP,RP,MN,P1,P2,E1)) = CP .
eq creator1(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = CP .
eq creator2(m3(CS,SS,RP,MN,E31,E32)) = CS .
eq creator1(m4(CP,SP,RP,MN,E4)) = CP .
-- メッセージの見かけの送信者
eq sender1(m1(CP,SP,RP,MN,P1,P2,E1)) = SP .
eq sender1(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = SP .
eq sender2(m3(CS,SS,RP,MN,E31,E32)) = SS .
eq sender1(m4(CP,SP,RP,MN,E4)) = SP .
-- メッセージの受信者
eq reciver1(m1(CP,SP,RP,MN,P1,P2,E1)) = RP .
eq reciver2(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = RS .
eq reciver1(m3(CS,SS,RP,MN,E31,E32)) = RP .
eq reciver1(m4(CP,SP,RP,MN,E4)) = RP .
-- ノンス M
eq mnonce(m1(CP,SP,RP,MN,P1,P2,E1)) = MN .
eq mnonce(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = MN .
eq mnonce(m3(CS,SS,RP,MN,E31,E32)) = MN .
eq mnonce(m4(CP,SP,RP,MN,E4)) = MN .
-- 主体の識別子
eq id1(m1(CP,SP,RP,MN,P1,P2,E1)) = P1 .
eq id2(m1(CP,SP,RP,MN,P1,P2,E1)) = P2 .
eq id1(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = P1 .
eq id2(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = P2 .
-- message 中の暗号文
eq ci1(m1(CP,SP,RP,MN,P1,P2,E1)) = E1 .
eq ci2a(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = E21 .
eq ci2b(m2(CP,SP,RS,MN,P1,P2,E21,E22)) = E22 .
eq ci3a(m3(CS,SS,RP,MN,E31,E32)) = E31 .
eq ci3b(m3(CS,SS,RP,MN,E31,E32)) = E32 .
eq ci4(m4(CP,SP,RP,MN,E4)) = E4 .
--
eq (M = M) = true .
ceq (M1 = M2) = (m1?(M2) and
 creator1(M1) = creator1(M2) and
 sender1(M1) = sender1(M2) and
 reciver1(M1) = reciver1(M2) and
 mnonce(M1) = mnonce(M2) and
 id1(M1) = id1(M2) and
 id2(M1) = id2(M2) and
 ci1(M1) = ci1(M2))
if m1?(M1) .
ceq (M1 = M2) = (m2?(M2) and
 creator1(M1) = creator1(M2) and
 sender1(M1) = sender1(M2) and
 reciver2(M1) = reciver2(M2) and
 mnonce(M1) = mnonce(M2) and
 id1(M1) = id1(M2) and
 id2(M1) = id2(M2) and
 ci2a(M1) = ci2a(M2) and
 ci2b(M1) = ci2b(M2))
if m2?(M1) .
ceq (M1 = M2) = (m3?(M2) and
 creator2(M1) = creator2(M2) and
 sender2(M1) = sender2(M2) and
 reciver1(M1) = reciver1(M2) and
 mnonce(M1) = mnonce(M2) and
 ci3a(M1) = ci3a(M2) and
 ci3b(M1) = ci3b(M2))
if m3?(M1) .
```

```
ceq (M1 = M2) = (m4?(M2) and
 creator1(M1) = creator1(M2) and
 sender1(M1) = sender1(M2) and
 reciver1(M1) = reciver1(M2) and
 mnonce(M1) = mnonce(M2) and
 ci4(M1) = ci4(M2))
if m4?(M1) .
}


--
-- パラメタつきモジュール宣言に必要なモジュール
--
mod* EQTRIV principal-sort Elt {
pr(TRIV)
op _=_ : Elt Elt -> Bool {comm}
}


--
-- 汎用の多重集合の定義
--
mod! BAG (D :: EQTRIV) principal-sort Bag {
[Elt.D < Bag]
op void : -> Bag
op _,_ : Bag Bag -> Bag {assoc comm id: void}
op _\in_ : Elt.D Bag -> Bool
var B : Bag
vars E1 E2 : Elt.D
eq E1 \in void = false .
ceq E1 \in (E2,B) = true if E1 = E2 .
ceq E1 \in (E2,B) = E1 \in B if not(E1 = E2) .
}


--
-- 汎用の集合の定義
--
mod! SET (D :: EQTRIV) principal-sort Set {
[Elt.D < Set]
op empty : -> Set
op __ : Set Set -> Set {assoc comm idem id: empty}
op _\in_ : Elt.D Set -> Bool
var S : Set
vars E1 E2 : Elt.D
eq E1 \in empty = false .
ceq E1 \in (E2 S) = true if E1 = E2 .
ceq E1 \in (E2 S) = E1 \in S if not(E1 = E2) .
}


--
-- 汎用のものの集まりを定義
--
mod* COLLECTION (D :: TRIV) principal-sort Col {
[Elt.D < Col]
op _\in_ : Elt.D Col -> Bool
}


--
-- ネットワークの定義
--
mod! NETWORK {
pr(PRINCIPAL + SERVER + MNONCE + NONCE + KEY)
pr(CIPHER1 + CIPHER2)
```

```
pr(BAG(MSG)*{sort Bag -> Network})
pr(COLLECTION(MNONCE)*{sort Col -> ColMnonce})
pr(COLLECTION(NONCE)*{sort Col -> ColNonce})
pr(COLLECTION(CIPHER1)*{sort Col -> ColCi1})
pr(COLLECTION(CIPHER2)*{sort Col -> ColCi2})
pr(COLLECTION(KEY)*{sort Col -> ColKey})
--
-- ネットワークから侵入者が取得できるノンス、Message1〜4 , 発行する共通鍵それぞれを表す
--op cmnonce : Network -> ColMnonce
op cnonce : Network -> ColNonce
op cci1   : Network -> ColCi1
op cci2a  : Network -> ColCi1
op cci2b  : Network -> ColCi1
op cci3a  : Network -> ColCi2
op cci3b  : Network -> ColCi2
op cci4   : Network -> ColCi2
op ckey : Network -> ColKey
--
var MN : Mnonce
var N : Nonce
vars E1 E21 E22 : Cipher1
vars E31 E32 E4 : Cipher2
var M : Msg
var NW : Network
var K : Key
--

-- cnonce 侵入者がネットワークから収集できるノンス N
-- 最初の等式　侵入者が生成したノンス N は侵入者が利用できる
-- 2,3,4,5,6,7 番目の等式 ネットワークにあるメッセージの暗号文が侵入者と認証局の共通鍵で
--  暗号化されていれば暗号文中のノンス N を利用できる
-- 8 番目の等式 どの条件も満たさない場合、そのメッセージは利用できない
eq N \in cnonce(void) = (creator(N) = intruder) .
ceq N \in cnonce(M,NW) = true if m1?(M) and pr0(skey(ci1(M))) = intruder .
ceq N \in cnonce(M,NW) = true if m2?(M) and pr0(skey(ci2a(M))) = intruder .
ceq N \in cnonce(M,NW) = true if m2?(M) and pr0(skey(ci2b(M))) = intruder .
ceq N \in cnonce(M,NW) = true if m3?(M) and pr0(skey(ci3a(M))) = intruder .
ceq N \in cnonce(M,NW) = true if m3?(M) and pr0(skey(ci3b(M))) = intruder .
ceq N \in cnonce(M,NW) = true if m4?(M) and pr0(skey(ci4(M))) = intruder .
ceq N \in cnonce(M,NW) = N \in cnonce(NW)
if not(m1?(M) and pr0(skey(ci1(M))) = intruder) and
 not(m2?(M) and pr0(skey(ci2a(M))) = intruder) and
 not(m2?(M) and pr0(skey(ci2b(M))) = intruder) and
 not(m3?(M) and pr0(skey(ci3a(M))) = intruder) and
 not(m3?(M) and pr0(skey(ci3b(M))) = intruder) and
 not(m4?(M) and pr0(skey(ci4(M))) = intruder) .


-- cci1 侵入者がネットワークから収集できるメッセージ 1 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message1 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message1 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message1 以外のメッセージからは該当する暗号文は得られない
--
eq E1 \in cci1(void) = false .
ceq E1 \in cci1(M,NW) = true
if m1?(M) and not(pr0(skey(ci1(M))) = intruder) and ci1(M) = E1 .
ceq E1 \in cci1(M,NW) = E1 \in cci1(NW)
if not(m1?(M) and not(pr0(skey(ci1(M))) = intruder) and ci1(M) = E1) .
--
-- cci2a 侵入者がネットワークから収集できるメッセージ 2 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message2 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message2 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message2 以外のメッセージからは該当する暗号文は得られない
--
eq E21 \in cci2a(void) = false .
ceq E21 \in cci2a(M,NW) = true
```

```
if m2?(M) and not(pr0(skey(ci2a(M))) = intruder) and ci2a(M) = E21 .
ceq E21 \in cci2a(M,NW) = E21 \in cci2a(NW)
if not(m2?(M) and not(pr0(skey(ci2a(M))) = intruder) and ci2a(M) = E21) .
--
-- cci2b 侵入者がネットワークから収集できるメッセージ 2 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message2 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message2 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message2 以外のメッセージからは該当する暗号文は得られない
--
eq E22 \in cci2b(void) = false .
ceq E22 \in cci2b(M,NW) = true
if m2?(M) and not(pr0(skey(ci2b(M))) = intruder) and ci2b(M) = E22 .
ceq E22 \in cci2b(M,NW) = E22 \in cci2b(NW)
if not(m2?(M) and not(pr0(skey(ci2b(M))) = intruder) and ci2b(M) = E22) .
--
-- cci3a 侵入者がネットワークから収集できるメッセージ 3 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message3 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message3 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message3 以外のメッセージからは該当する暗号文は得られない
--
eq E31 \in cci3a(void) = false .
ceq E31 \in cci3a(M,NW) = true
if m3?(M) and not(pr0(skey(ci3a(M))) = intruder) and ci3a(M) = E31 .
ceq E31 \in cci3a(M,NW) = E31 \in cci3a(NW)
if not(m3?(M) and not(pr0(skey(ci3a(M))) = intruder) and ci3a(M) = E31) .
--
-- cci3b 侵入者がネットワークから収集できるメッセージ 3 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message3 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message3 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message3 以外のメッセージからは該当する暗号文は得られない
--
eq E32 \in cci3b(void) = false .
ceq E32 \in cci3b(M,NW) = true
if m3?(M) and not(pr0(skey(ci3b(M))) = intruder) and ci3b(M) = E32 .
ceq E32 \in cci3b(M,NW) = E32 \in cci3b(NW)
if not(m3?(M) and not(pr0(skey(ci3b(M))) = intruder) and ci3b(M) = E32) .
--
-- cci4 侵入者がネットワークから収集できるメッセージ 4 中の暗号文
-- 1 番目 ネットワークが空であれば、侵入者が利用できる Message4 に現れる暗号文は無い
-- 2 番目 ネットワーク中に Message4 が存在すれば、その中に現れる暗号文を利用できる
-- 3 番目 Message4 以外のメッセージからは該当する暗号文は得られない
--
eq E4 \in cci4(void) = false .
ceq E4 \in cci4(M,NW) = true
if m4?(M) and not(pr0(skey(ci4(M))) = intruder) and ci4(M) = E4 .
ceq E4 \in cci4(M,NW) = E4 \in cci4(NW)
if not(m4?(M) and not(pr0(skey(ci4(M))) = intruder) and ci4(M) = E4) .


-- ckey 侵入者がネットワークから収集できる共通鍵
-- 1 番目   ネットワークが空であれば、侵入者が利用できる共通鍵はない
-- 2,3,4 番目 ネットワーク中のメッセージの暗号文が、
--侵入者と認証局との共通鍵で暗号化されていれば暗号文中の共通鍵を利用できる
-- 5 番目   1〜4 番目のどの条件も満たさない場合、そのメッセージは利用できない
eq K \in ckey(void) = false .
ceq K \in ckey(M,NW) = true
if m3?(M) and pr0(skey(ci3a(M))) = intruder and key(ci3a(M)) = K .
ceq K \in ckey(M,NW) = true
if m3?(M) and pr0(skey(ci3b(M))) = intruder and key(ci3b(M)) = K .
ceq K \in ckey(M,NW) = true
if m4?(M) and pr0(skey(ci4(M))) = intruder and key(ci4(M)) = K .
ceq K \in ckey(M,NW) = K \in ckey(NW)
 if not(m3?(M) and pr0(skey(ci3a(M))) = intruder and key(ci3a(M)) = K) and
  not(m3?(M) and pr0(skey(ci3b(M))) = intruder and key(ci3b(M)) = K) and
  not(m4?(M) and pr0(skey(ci4(M))) = intruder and key(ci4(M)) = K) .
```

```
}

------------------------------------------------------------------------------
-- OTWAY-REES Protocol のモデル
--
mod* OTWAY-REES {
pr(NETWORK)
pr(SET(MVALUE)*{sort Set -> Umvalue})
pr(SET(NVALUE)*{sort Set -> Uvalue})
pr(SET(PUBKEY)*{sort Set -> Sharekey})
*[System]*
-- any initial state
op init : -> System

-- observation operations
bop uv : System -> Uvalue
bop um : System -> Umvalue
bop nw : System -> Network
bop sk : System -> Sharekey

-- action operations
-- sending messges
--  M1 送信者  M1 受信者  ノンス M ノンス Na
bop mes1 : System Principal Principal Mvalue Nvalue      -> System
--  M2 送信者  ノンス Nb message1
bop mes2 : System Principal Nvalue Msg  -> System
--  messgae2 生成する共通鍵
bop mes3 : System Msg Pubkey -> System
--  M4 送信者  message1 messgae2 messgage3
bop mes4 : System Principal Msg Msg Msg -> System

-- faking messages
-- for message1
bop fkm11 : System Principal Principal Mnonce Cipher1 -> System
bop fkm12 : System Principal Principal Mnonce Nonce Skey -> System

-- for message2
bop fkm21 : System Principal Principal Mnonce Cipher1 Cipher1 -> System
bop fkm22 : System Principal Principal Mnonce Cipher1 Nonce Skey -> System
bop fkm23 : System Principal Principal Mnonce Cipher1 Nonce Skey -> System
bop fkm24 : System Principal Principal Mnonce Nonce Nonce Skey Skey -> System

-- for message3
bop fkm31 : System Principal Principal Mnonce Cipher2 Cipher2 -> System
bop fkm32 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> System
bop fkm33 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> System
bop fkm34 : System Principal Principal Mnonce Nonce Nonce Skey Skey Key -> System
-- for message4
bop fkm41 : System Principal Principal Mnonce Cipher2 -> System
bop fkm42 : System Principal Principal Mnonce Nonce Skey Key -> System

-- CafeOBJ variables
var S : System
vars M1 M2 M3 : Msg
vars P Q : Principal
var MV : Mvalue
vars V1 V2 : Nvalue
var PK : Pubkey
vars MN : Mnonce
vars N1 N2 : Nonce
vars E11 E12 : Cipher1
vars E21 E22 : Cipher2
var K : Key
vars K1 K2 : Skey

-- for any initial state
```

```
eq um(init) = empty .
eq uv(init) = empty .
eq nw(init) = void .
eq sk(init) = empty .



-- for mes1
-- 効力条件 生成するノンス M,Na がそれぞれ未使用
-- 事後状態 ネットワークにメッセージを追加、
--  um,uv で観測できる集合に生成したノンス M,Na をそれぞれ追加
--
op c-mes1 : System Principal Principal Mvalue Nvalue -> Bool
eq c-mes1(S,P,Q,MV,V1) = not(MV \in um(S)) and not(V1 \in uv(S)) .
--
ceq uv(mes1(S,P,Q,MV,V1)) = V1 uv(S) if c-mes1(S,P,Q,MV,V1) .
ceq um(mes1(S,P,Q,MV,V1)) = MV um(S) if c-mes1(S,P,Q,MV,V1) .
ceq nw(mes1(S,P,Q,MV,V1))
= m1(P,P,Q,mnonce(P,Q,MV),P,Q,enc1(skey(P),mnonce(P,Q,MV),nonce(P,Q,V1),P,Q)) , nw(S)
if c-mes1(S,P,Q,MV,V1) .
eq sk(mes1(S,P,Q,MV,V1)) = sk(S) .
ceq mes1(S,P,Q,MV,V1) = S if not c-mes1(S,P,Q,MV,V1) .



-- for mes2
-- 効力条件 ある条件を満たす Message1 が存在、生成するノンス Nb が未使用
--  Message1 の条件 受信者が Q
-- 事後状態 ネットワークにメッセージを追加、uv で観測できる集合に生成したノンスを追加
--
op c-mes2 : System Principal Nvalue Msg -> Bool
eq c-mes2(S,Q,V2,M1) = M1 \in nw(S) and
 m1?(M1) and
 reciver1(M1) = Q and
 not(V2 \in uv(S)) .
--
eq um(mes2(S,Q,V2,M1)) = um(S) .
ceq uv(mes2(S,Q,V2,M1)) = V2 uv(S) if c-mes2(S,Q,V2,M1) .
ceq nw(mes2(S,Q,V2,M1))
= m2(Q,Q,ca,mnonce(M1),id1(M1),Q,
 ci1(M1),enc1(skey(Q),mnonce(M1),nonce(Q,id1(M1),V2),id1(M1),id2(M1))) , nw(S)
if c-mes2(S,Q,V2,M1) .
eq sk(mes2(S,Q,V2,M1)) = sk(S) .
ceq mes2(S,Q,V2,M1) = S if not c-mes2(S,Q,V2,M1) .



-- for mes3
-- 効力条件 ある条件を満たす Message2 が存在、生成する共通鍵が未使用
--  Message2 の条件 受信者が CA,
--メッセージ内の暗号文が A と CA の共通鍵,B と CA の共通鍵でそれぞれ暗号化、
--暗号文中のノンス M, 認証子 A,B がメッセージ内のノンス M, 認証子 A,B と一致
-- 事後状態 ネットワークにメッセージを追加、生成した共通鍵を共通鍵の集合に追加
--
op c-mes3 : System Msg Pubkey -> Bool
eq c-mes3(S,M2,PK) = M2 \in nw(S) and m2?(M2) and
 reciver2(M2) = ca and
 pr0(skey(ci2a(M2))) = id1(M2) and
 pr0(skey(ci2b(M2))) = id2(M2) and
 mnonce(M2) = mnonce(ci2a(M2)) and
 mnonce(M2) = mnonce(ci2b(M2)) and
 mnonce(ci2a(M2)) = mnonce(ci2b(M2)) and
 p(ci2a(M2)) = id1(M2) and
 q(ci2a(M2)) = id2(M2) and
 p(ci2b(M2)) = id1(M2) and
 q(ci2b(M2)) = id2(M2) and
```

```
 not(PK \in sk(S)) .
--
eq um(mes3(S,M2,PK)) = um(S) .
eq uv(mes3(S,M2,PK)) = uv(S) .
ceq nw(mes3(S,M2,PK))
= m3(reciver2(M2),reciver2(M2),sender1(M2),mnonce(M2),
 enc2(skey(id1(M2)),nonce(ci2a(M2)),key(id1(M2),id2(M2),PK)),
 enc2(skey(id2(M2)),nonce(ci2b(M2)),key(id1(M2),id2(M2),PK))) , nw(S)
if c-mes3(S,M2,PK) .
ceq sk(mes3(S,M2,PK)) = PK sk(S)
  if c-mes3(S,M2,PK) .
ceq mes3(S,M2,PK) = S if not c-mes3(S,M2,PK) .



-- for mes4
-- 効力条件 ある条件を満たす Message1,Message2,Message3 が存在
--  Message1 の条件 受信者が B
--  Message2 の条件 見掛けの送信者が B
--受信者が CA, メッセージ内の Message1 以外の暗号文が Q と CA の共通鍵で暗号化、
--その暗号文中のノンス M, 認証子 A,B がメッセージ内のノンス M, 認証子 A,B と一致
--Message1 のノンス M と Message2 のノンス M が一致
--Message3 の条件 見掛けの送信者が CA、受信者が B
--暗号文のうち 1 つの暗号鍵が主体 B と認証局との共通鍵である
--ノンス M が Message1 のノンス M と一致する,Message2 で生成したノンスが暗号文中に含まれている
-- 事後状態 ネットワークにメッセージを追加
--
op c-mes4 : System Principal Msg Msg Msg -> Bool
eq c-mes4(S,Q,M1,M2,M3) = M1 \in nw(S) and m1?(M1) and
M2 \in nw(S) and m2?(M2) and
M3 \in nw(S) and m3?(M3) and
reciver1(M1) = Q and
sender1(M2) = Q and
reciver2(M2) = ca and
pr0(skey(ci2b(M2))) = Q and
mnonce(M1) = mnonce(M2) and
mnonce(M2) = mnonce(ci2b(M2)) and
id2(M2) = Q and
p(ci2b(M2)) = id1(M2) and
q(ci2b(M2)) = id2(M2) and
sender2(M3) = ca and
reciver1(M3) = Q and
pr0(skey(ci3b(M3))) = Q and
mnonce(M3) = mnonce(M2) and
nonce(ci3b(M3)) = nonce(ci2b(M2)) .
--
eq um(mes4(S,Q,M1,M2,M3)) = um(S) .
eq uv(mes4(S,Q,M1,M2,M3)) = uv(S) .
ceq nw(mes4(S,Q,M1,M2,M3))
= m4(Q,Q,sender1(M1),mnonce(M3),ci3a(M3)) , nw(S)
if c-mes4(S,Q,M1,M2,M3) .
eq sk(mes4(S,Q,M1,M2,M3)) = sk(S) .
ceq mes4(S,Q,M1,M2,M3) = S if not c-mes4(S,Q,M1,M2,M3) .



-- for action fkm11
-- 効力条件 メッセージを捏造するのに必要な暗号文 C1 を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm11 : System Principal Principal Mnonce Cipher1 -> Bool
eq c-fkm11(S,P,Q,MN,E11) = E11 \in cci1(nw(S)) .
--
eq uv(fkm11(S,P,Q,MN,E11)) = uv(S) .
ceq nw(fkm11(S,P,Q,MN,E11)) = m1(intruder,P,Q,MN,P,Q,E11) , nw(S)
if c-fkm11(S,P,Q,MN,E11) .
eq sk(fkm11(S,P,Q,MN,E11)) = sk(S) .
```

```
ceq fkm11(S,P,Q,MN,E11) = S if not c-fkm11(S,P,Q,MN,E11) .


-- for action fkm12
-- 効力条件 メッセージを捏造するのに必要なノンス Np、共通鍵 Kp を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm12 : System Principal Principal Mnonce Nonce Skey -> Bool
eq c-fkm12(S,P,Q,MN,N1,K1) = N1 \in cnonce(nw(S)) and
 pr0(K1) = intruder .
--
eq uv(fkm12(S,P,Q,MN,N1,K1)) = uv(S) .
ceq nw(fkm12(S,P,Q,MN,N1,K1)) = m1(intruder,P,Q,MN,P,Q,enc1(K1,MN,N1,P,Q)) , nw(S)
  if c-fkm12(S,P,Q,MN,N1,K1) .
eq sk(fkm12(S,P,Q,MN,N1,K1)) = sk(S) .
ceq fkm12(S,P,Q,MN,N1,K1) = S if not(c-fkm12(S,P,Q,MN,N1,K1)) .


-- for action fkm21
-- 効力条件 メッセージを捏造するのに必要な暗号文 C1,C2 を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm21 : System Principal Principal Mnonce Cipher1 Cipher1 -> Bool
eq c-fkm21(S,P,Q,MN,E11,E12) = E11 \in cci2a(nw(S)) and
 E12 \in cci2b(nw(S)) .
--
eq uv(fkm21(S,P,Q,MN,E11,E12)) = uv(S) .
ceq nw(fkm21(S,P,Q,MN,E11,E12)) = m2(intruder,P,ca,MN,P,Q,E11,E12) , nw(S)
  if c-fkm21(S,P,Q,MN,E11,E12) .
eq sk(fkm21(S,P,Q,MN,E11,E12)) = sk(S) .
ceq fkm21(S,P,Q,MN,E11,E12) = S if not c-fkm21(S,P,Q,MN,E11,E12) .


-- for action fkm22
-- 効力条件 メッセージを捏造するのに必要な暗号文 C1、ノンス Nb、共通鍵 Kq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm22 : System Principal Principal Mnonce Cipher1 Nonce Skey -> Bool
eq c-fkm22(S,P,Q,MN,E11,N2,K2) = E11 \in cci2a(nw(S)) and
   N2 \in cnonce(nw(S)) and
   pr0(K2) = intruder .
--
eq uv(fkm22(S,P,Q,MN,E11,N2,K2)) = uv(S) .
ceq nw(fkm22(S,P,Q,MN,E11,N2,K2)) = m2(intruder,Q,ca,MN,P,Q,E11,enc1(K2,MN,N2,P,Q)) , nw(S)
  if c-fkm22(S,P,Q,MN,E11,N2,K2) .
eq sk(fkm22(S,P,Q,MN,E11,N2,K2)) = sk(S) .
ceq fkm22(S,P,Q,MN,E11,N2,K2) = S if not c-fkm22(S,P,Q,MN,E11,N2,K2) .


-- for action fkm23
-- 効力条件 メッセージを捏造するのに必要な暗号文 C2、ノンス Na、共通鍵 Kp を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm23 : System Principal Principal Mnonce Cipher1 Nonce Skey -> Bool
eq c-fkm23(S,P,Q,MN,E12,N1,K1) = E12 \in cci2b(nw(S)) and
   N1 \in cnonce(nw(S)) and
   pr0(K1) = intruder .
--
eq uv(fkm23(S,P,Q,MN,E12,N1,K1)) = uv(S) .
ceq nw(fkm23(S,P,Q,MN,E12,N1,K1)) = m2(intruder,Q,ca,MN,P,Q,enc1(K1,MN,N1,P,Q),E12) , nw(S)
  if c-fkm23(S,P,Q,MN,E12,N1,K1) .
eq sk(fkm23(S,P,Q,MN,E12,N1,K1)) = sk(S) .
ceq fkm23(S,P,Q,MN,E12,N1,K1) = S if not c-fkm23(S,P,Q,MN,E12,N1,K1) .


-- for action fkm24
-- 効力条件 メッセージを捏造するのに必要なノンス Np,Nq、共通鍵 Kp,Kq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm24 : System Principal Principal Mnonce Nonce Nonce Skey Skey -> Bool
eq c-fkm24(S,P,Q,MN,N1,N2,K1,K2) = N1 \in cnonce(nw(S)) and
```

```
   N2 \in cnonce(nw(S)) and
   pr0(K1) = intruder and
   pr0(K2) = intruder .
--
eq uv(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = uv(S) .
ceq nw(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = m2(intruder,Q,ca,MN,P,Q,enc1(K1,MN,N1,P,Q),enc1(K2,MN,N2,P,Q)) , nw(S)
if c-fkm24(S,P,Q,MN,N1,N2,K1,K2) .
eq sk(fkm24(S,P,Q,MN,N1,N2,K1,K2)) = sk(S) .
ceq fkm24(S,P,Q,MN,N1,N2,K1,K2) = S if not c-fkm24(S,P,Q,MN,N1,N2,K1,K2) .


-- for action fkm31
-- 効力条件 メッセージを捏造するのに必要な暗号文を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm31 : System Principal Principal Mnonce Cipher2 Cipher2 -> Bool
eq c-fkm31(S,P,Q,MN,E21,E22) = E21 \in cci3a(nw(S)) and
 E22 \in cci3b(nw(S)) .
--
eq uv(fkm31(S,P,Q,MN,E21,E22)) = uv(S) .
ceq nw(fkm31(S,P,Q,MN,E21,E22)) = m3(intruder,ca,P,MN,E21,E22) , nw(S)
if c-fkm31(S,P,Q,MN,E21,E22) .
eq sk(fkm31(S,P,Q,MN,E21,E22)) = sk(S) .
ceq fkm31(S,P,Q,MN,E21,E22) = S if not c-fkm31(S,P,Q,MN,E21,E22) .


-- for action fkm32
-- 効力条件 メッセージを捏造するのに必要な暗号文 C3, ノンス Nq, 共通鍵 Kq,Kpq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm32 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> Bool
eq c-fkm32(S,P,Q,MN,E21,N2,K2,K) = E21 \in cci3a(nw(S)) and
   N2 \in cnonce(nw(S)) and
     pr0(K2) = intruder and
     K \in ckey(nw(S)) .
--
eq uv(fkm32(S,P,Q,MN,E21,N2,K2,K)) = uv(S) .
ceq nw(fkm32(S,P,Q,MN,E21,N2,K2,K)) = m3(intruder,ca,Q,MN,E21,enc2(K2,N2,K)) , nw(S)
  if c-fkm32(S,P,Q,MN,E21,N2,K2,K) .
eq sk(fkm32(S,P,Q,MN,E21,N2,K2,K)) = sk(S) .
ceq fkm32(S,P,Q,MN,E21,N2,K2,K) = S if not c-fkm32(S,P,Q,MN,E21,N2,K2,K) .


-- for action fkm33
-- 効力条件 メッセージを捏造するのに必要な暗号文 C4, ノンス Np, 共通鍵 Kp,Kpq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm33 : System Principal Principal Mnonce Cipher2 Nonce Skey Key -> Bool
eq c-fkm33(S,P,Q,MN,E22,N1,K1,K) = E22 \in cci3b(nw(S)) and
 N1 \in cnonce(nw(S)) and
     pr0(K1) = intruder and
     K \in ckey(nw(S)) .
--
eq uv(fkm33(S,P,Q,MN,E22,N1,K1,K)) = uv(S) .
ceq nw(fkm33(S,P,Q,MN,E22,N1,K1,K)) = m3(intruder,ca,Q,MN,enc2(K1,N1,K),E22) , nw(S)
if c-fkm33(S,P,Q,MN,E22,N1,K1,K) .
eq sk(fkm33(S,P,Q,MN,E22,N1,K1,K)) = sk(S) .
ceq fkm33(S,P,Q,MN,E22,N1,K1,K) = S if not c-fkm33(S,P,Q,MN,E22,N1,K1,K) .


-- for action fkm34
-- 効力条件 メッセージを捏造するのに必要なノンス Np,Nq、, 共通鍵 Kp,Kq,Kpq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm34 : System Principal Principal Mnonce Nonce Nonce Skey Skey Key -> Bool
eq c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) = N1 \in cnonce(nw(S)) and
   N2 \in cnonce(nw(S)) and
     pr0(K1) = intruder and
     pr0(K2) = intruder and
     K \in ckey(nw(S)) .
```

```
--
eq uv(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = uv(S) .
ceq nw(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = m3(intruder,ca,Q,MN,enc2(K1,N1,K),enc2(K2,N2,K)) , nw(S)
if c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) .
eq sk(fkm34(S,P,Q,MN,N1,N2,K1,K2,K)) = sk(S) .
ceq fkm34(S,P,Q,MN,N1,N2,K1,K2,K) = S if not c-fkm34(S,P,Q,MN,N1,N2,K1,K2,K) .


-- for actin fkm41
-- 効力条件 メッセージを捏造するのに必要な暗号文 C3 を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm41 : System Principal Principal Mnonce Cipher2 -> Bool
eq c-fkm41(S,P,Q,MN,E21) = E21 \in cci4(nw(S)) .
--
eq uv(fkm41(S,P,Q,MN,E21)) = uv(S) .
ceq nw(fkm41(S,P,Q,MN,E21)) = m4(intruder,Q,P,MN,E21) ,nw(S)
  if c-fkm41(S,P,Q,MN,E21) .
eq sk(fkm41(S,P,Q,MN,E21)) = sk(S) .
ceq fkm41(S,P,Q,MN,E21) = S if not c-fkm41(S,P,Q,MN,E21) .


-- for actino fkm42
-- 効力条件 メッセージを捏造するのに必要なノンス Np、共通鍵 Kp,Kpq を侵入者が取得している
-- 事後状態 捏造したメッセージをネットワークに追加
op c-fkm42 : System Principal Principal Mnonce Nonce Skey Key -> Bool
eq c-fkm42(S,P,Q,MN,N1,K1,K) = N1 \in cnonce(nw(S)) and
 pr0(K1) = intruder and
 K \in ckey(nw(S)) .
--
eq uv(fkm42(S,P,Q,MN,N1,K1,K)) = uv(S) .
ceq nw(fkm42(S,P,Q,MN,N1,K1,K)) = m4(intruder,Q,P,MN,enc2(K1,N1,K)) , nw(S)
  if c-fkm42(S,P,Q,MN,N1,K1,K) .
eq sk(fkm42(S,P,Q,MN,N1,K1,K)) = sk(S) .
ceq fkm42(S,P,Q,MN,N1,K1,K) = S if not c-fkm42(S,P,Q,MN,N1,K1,K) .
}
```

付 録B　検証したい性質の記述

```
------------------------------------------------------
-- Invariant
-- Begin Secrecy Theorem
--
------------------------------------------------------

mod INV {
pr(OTWAY-REES)
--
ops e1 e21 e22 : -> Cipher1
ops e31 e32 e4 : -> Cipher2
op k : -> Key

-- declare invariants to prove
op inv010 : System Cipher1 -> Bool
op inv020 : System Cipher1 -> Bool
op inv030 : System Cipher1 -> Bool
op inv040 : System Cipher2 -> Bool
op inv050 : System Cipher2 -> Bool
op inv060 : System Cipher2 -> Bool
op inv100 : System Key -> Bool

-- CafeOBJ variables
var S : System
vars E1 E21 E22 : Cipher1
vars E31 E32 E4 : Cipher2
var K : Key

-- define invariants to prove
eq inv010(S,E1) = (E1 \in cci1(nw(S)) implies not(pr0(skey(E1)) = intruder)) .
eq inv020(S,E21) = (E21 \in cci2a(nw(S)) implies not(pr0(skey(E21)) = intruder)) .
eq inv030(S,E22) = (E22 \in cci2b(nw(S)) implies not(pr0(skey(E22)) = intruder)) .
eq inv040(S,E31) = (E31 \in cci3a(nw(S)) implies not(pr0(skey(E31)) = intruder)) .
eq inv050(S,E32) = (E32 \in cci3b(nw(S)) implies not(pr0(skey(E32)) = intruder)) .
eq inv060(S,E4) = (E4 \in cci4(nw(S)) implies not(pr0(skey(E4)) = intruder)) .
eq inv100(S,K) = (K \in ckey(nw(S)) implies (pr1(K) = intruder or pr2(K) = intruder)) .

}

mod ISTEP {
pr(INV)
-- arbitrary objects
ops s s' : -> System

-- declare predicates to prove in inductive step
op istep010 : Cipher1 -> Bool
op istep020 : Cipher1 -> Bool
op istep030 : Cipher1 -> Bool
op istep040 : Cipher2 -> Bool
op istep050 : Cipher2 -> Bool
op istep060 : Cipher2 -> Bool
op istep100 : Key -> Bool

-- CafeOBJ variables
```

```
vars E1 E21 E22 : Cipher1
vars E31 E32 E4 : Cipher2
var K : Key

-- define predicates to prove in inductive step
eq istep010(E1) = inv010(s,E1) implies inv010(s',E1) .
eq istep020(E21) = inv020(s,E21) implies inv020(s',E21) .
eq istep030(E22) = inv030(s,E22) implies inv030(s',E22) .
eq istep040(E31) = inv040(s,E31) implies inv040(s',E31) .
eq istep050(E32) = inv050(s,E32) implies inv050(s',E32) .
eq istep060(E4) = inv060(s,E4) implies inv060(s',E4) .
eq istep100(K) = inv100(s,K) implies inv100(s',K) .

}
```

付 録C　表明1の証明譜

```
--> ********************************************************************************** --
-- I) Base case

open INV
red inv100(init,k) .
close



-- II) Inductive case
--> ********************************************************************************** --
--> 1) mes1(s,p10,q10,mv10,r10) --
-- 1.1) c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
-- eq c-mes1(s,p10,q10,mv10,r10) = true .
eq mv10 \in um(s) = false .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predicate is true
red istep100(k) .
close


-- 1.2) not c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
eq c-mes1(s,p10,q10,mv10,r10) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predivate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 2) mes2(s,q10,r10,mes10) --
-- 2.1) c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary object
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes2(s,q10,r10,mes10) = true .
eq nw(s) = mes10 , nw10 .
eq m1?(mes10) = true .
eq reciver1(mes10) = q10 .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep100(k) .
close


-- 2.2) not c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
-- assumptions
 eq c-mes2(s,q10,r10,mes10) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 3) mes3(s,mes10,pk10) --
-- 3.1) c-mes(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq (k = key(id1(mes10),id2(mes10),pk10)) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq k = key(id1(mes10),id2(mes10),pk10) .
eq id1(mes10) = intruder .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq k = key(id1(mes10),id2(mes10),pk10) .
eq (id1(mes10) = intruder) = false .
eq id2(mes10) = intruder .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
```

```
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq k = key(id1(mes10),id2(mes10),pk10) .
eq (id1(mes10) = intruder) = false .
eq (id2(mes10) = intruder) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep100(k) .
close


-- 3.2) not c-mes3(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
-- assumptions
eq c-mes3(s,mes10,pk10) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 4) mes4(s,q10,mes10,mes20,mes30) --
-- 4.1) c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq (k = key(ci3a(mes30))) = false .
```

```
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq k = key(ci3a(mes30)) .
eq pr0(skey(ci3a(mes30))) = intruder .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
```

```
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq k = key(ci3a(mes30)) .
eq (pr0(skey(ci3a(mes30))) = intruder) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep100(k) .
close


-- 4.2) not c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
-- assumptions
eq c-mes4(s,q10,mes10,mes20,mes30) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep100(k) .
close


--> ******************************************************************************** --
--> 5) fkm11(s,p10,q10,m10,e10) --
-- 5.1) c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
--eq c-fkm11(s,p10,q10,m10,e10) = true .
eq e10 \in cci1(nw(s)) = true .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep100(k) .
close


-- 5.2) not c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
eq c-fkm11(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep100(k) .
close



--> ******************************************************************************** --
--> 6) fkm12(s,p10,q10,m10,n10,k10) --
-- 6.1) c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm12(s,p10,q10,m10,n10,k10) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder  .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep100(k) .
close


-- 6.2) not c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm12(s,p10,q10,m10,n10,k10) = false .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 7) fkm21(s,p10,q10,m10,e10,e20) --
-- 7.1) c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
-- eq c-fkm21(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq e20 \in cci2b(nw(s)) = true .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep100(k) .
close


-- 7.2) not c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
eq c-fkm21(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
```

```
--> 8) fkm22(s,p10,q10,m10,e10,n20,k20) --
-- 8.1) c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
-- eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep100(k) .
close


-- 8.2) not c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = false .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 9) fkm23(s,p10,q10,m10,e20,n10,k10) --
-- 9.1) c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = true .
eq e20 \in cci2b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep100(k) .
close


-- 9.2) not c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = false .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 10) fkm24(s,p10,q10,m10,n10,n20,k10,k20) --
-- 10.1) c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
-- eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep100(k) .
close


-- 10.2) not c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = false .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 11) fkm31(s,p10,q10,m10,e10,e20) --
-- 11.1) c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq pr0(skey(e10)) = intruder .
```

```
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red inv040(s,e10) implies istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq (pr0(skey(e10)) = intruder) = false .
eq pr0(skey(e20)) = intruder .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red inv050(s,e20) implies istep100(k) .
--red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq (pr0(skey(e10)) = intruder) = false .
eq (pr0(skey(e20)) = intruder) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep100(k) .
close


-- 11.2) not c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
eq c-fkm31(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************* --
--> 12) fkm32(s,p10,q10,m10,e10,n20,k20,k30) --
-- 12.1) c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
```

```
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq k = k30 .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
eq (pr0(skey(e10)) = intruder) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
eq pr0(skey(e10)) = intruder .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
```

```
red inv040(s,e10) implies istep100(k) .
close


-- 12.2) not c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 13) fkm33(s,p10,q10,m10,e20,n10,k10,k30) --
-- 13.1) c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq k = k30 .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
eq (pr0(skey(e20)) = intruder) = false .
```

```
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
eq pr0(skey(e20)) = intruder .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red inv050(s,e20) implies istep100(k) .
close


-- 13.2) not c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 14) fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) --
-- 14.1) c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
```

```
eq k30 \in ckey(nw(s)) = true .
--
eq k = k30 .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close


-- 14.2) not c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = false .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 15) fkm41(s,p10,q10,m10,e10) --
-- 15.1) c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
eq e10 \in cci4(nw(s)) = true .
--
eq (pr0(skey(e10)) = intruder) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
```

```
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
eq e10 \in cci4(nw(s)) = true .
--
eq pr0(skey(e10)) = intruder .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red inv060(s,e10) implies istep100(k) .
close


-- 15.2) not c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
eq c-fkm41(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep100(k) .
close



--> ********************************************************************************** --
--> 16) fkm42(s,p10,q10,m10,n10,k10,k30) --
-- 16.1) c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (k = k30) = false .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq k = k30 .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close


-- 16.2) not c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = false .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep100(k) .
close
```

付 録D　補題1の証明譜

```
--> ********************************************************************************** --
-- I) Base case

open INV
red inv040(init,e31) .
close



-- II) Inductive case
--> ********************************************************************************** --
--> 1) mes1(s,p10,q10,mv10,r10) --
-- 1.1) c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
-- eq c-mes1(s,p10,q10,mv10,r10) = true .
eq mv10 \in um(s) = false .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 1.2) not c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
eq c-mes1(s,p10,q10,mv10,r10) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predivate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 2) mes2(s,q10,r10,mes10) --
-- 2.1) c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary object
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes2(s,q10,r10,mes10) = true .
eq nw(s) = mes10 , nw10 .
eq m1?(mes10) = true .
eq reciver1(mes10) = q10 .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 2.2) not c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
-- assumptions
 eq c-mes2(s,q10,r10,mes10) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 3) mes3(s,mes10,pk10) --
-- 3.1) c-mes(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq (e31 = enc2(skey(id1(mes10)),nonce(ci2a(mes10)),key(id1(mes10),id2(mes10),pk10))) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep040(e31) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq e31 = enc2(skey(id1(mes10)),nonce(ci2a(mes10)),key(id1(mes10),id2(mes10),pk10)) .
eq id1(mes10) = intruder .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep040(e31) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq e31 = enc2(skey(id1(mes10)),nonce(ci2a(mes10)),key(id1(mes10),id2(mes10),pk10)) .
eq (id1(mes10) = intruder) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 3.2) not c-mes3(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
-- assumptions
eq c-mes3(s,mes10,pk10) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep040(e31) .
```

```
close


--> ********************************************************************************** --
--> 4) mes4(s,q10,mes10,mes20,mes30) --
-- 4.1) c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep040(e31) .
close


-- 4.2) not c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
-- assumptions
eq c-mes4(s,q10,mes10,mes20,mes30) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 5) fkm11(s,p10,q10,m10,e10) --
-- 5.1) c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
--eq c-fkm11(s,p10,q10,m10,e10) = true .
eq e10 \in cci1(nw(s)) = true .
-- successor state
```

```
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 5.2) not c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
eq c-fkm11(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 6) fkm12(s,p10,q10,m10,n10,k10) --
-- 6.1) c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm12(s,p10,q10,m10,n10,k10) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder  .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 6.2) not c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm12(s,p10,q10,m10,n10,k10) = false .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 7) fkm21(s,p10,q10,m10,e10,e20) --
-- 7.1) c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
ops e10 e20 : -> Cipher1 .
-- assumptions
-- eq c-fkm21(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq e20 \in cci2b(nw(s)) = true .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep040(e31) .
close


-- 7.2) not c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
eq c-fkm21(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 8) fkm22(s,p10,q10,m10,e10,n20,k20) --
-- 8.1) c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
-- eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep040(e31) .
close


-- 8.2) not c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = false .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep040(e31) .
close
```

```
--> ********************************************************************************** --
--> 9) fkm23(s,p10,q10,m10,e20,n10,k10) --
-- 9.1) c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = true .
eq e20 \in cci2b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 9.2) not c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = false .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 10) fkm24(s,p10,q10,m10,n10,n20,k10,k20) --
-- 10.1) c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
-- eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep040(e31) .
close


-- 10.2) not c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
```

```
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = false .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep040(e31) .
close



--> ******************************************************************************** --
--> 11) fkm31(s,p10,q10,m10,e10,e20) --
-- 11.1) c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq (e31 = e10) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep040(e31) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq e31 = e10 .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep040(e31) .
close


-- 11.2) not c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
eq c-fkm31(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep040(e31) .
```

```
close


--> ********************************************************************************** --
--> 12) fkm32(s,p10,q10,m10,e10,n20,k20,k30) --
-- 12.1) c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (e31 = e10) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep040(e31) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq e31 = e10 .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep040(e31) .
close


-- 12.2) not c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
```

```
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 13) fkm33(s,p10,q10,m10,e20,n10,k10,k30) --
-- 13.1) c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep040(e31) .
close


-- 13.2) not c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 14) fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) --
-- 14.1) c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
```

```
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep040(e31) .
close


-- 14.2) not c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = false .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 15) fkm41(s,p10,q10,m10,e10) --
-- 15.1) c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
eq e10 \in cci4(nw(s)) = true .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep040(e31) .
close


-- 15.2) not c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
eq c-fkm41(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep040(e31) .
close



--> ********************************************************************************** --
--> 16) fkm42(s,p10,q10,m10,n10,k10,k30) --
-- 16.1) c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep040(e31) .
close


-- 16.2) not c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = false .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep040(e31) .
close
```

付 録E　補題2の証明譜

```
--> ********************************************************************************** --
-- I) Base case

open INV
red inv050(init,e32) .
close



-- II) Inductive case
--> ********************************************************************************** --
--> 1) mes1(s,p10,q10,mv10,r10) --
-- 1.1) c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
-- eq c-mes1(s,p10,q10,mv10,r10) = true .
eq mv10 \in um(s) = false .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 1.2) not c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
eq c-mes1(s,p10,q10,mv10,r10) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predivate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 2) mes2(s,q10,r10,mes10) --
-- 2.1) c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary object
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes2(s,q10,r10,mes10) = true .
eq nw(s) = mes10 , nw10 .
eq m1?(mes10) = true .
eq reciver1(mes10) = q10 .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 2.2) not c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
-- assumptions
 eq c-mes2(s,q10,r10,mes10) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 3) mes3(s,mes10,pk10) --
-- 3.1) c-mes(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq (e32 = enc2(skey(id2(mes10)),nonce(ci2b(mes10)),key(id1(mes10),id2(mes10),pk10))) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep050(e32) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq e32 = enc2(skey(id2(mes10)),nonce(ci2b(mes10)),key(id1(mes10),id2(mes10),pk10)) .
eq id2(mes10) = intruder .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep050(e32) .
close


open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
--
eq e32 = enc2(skey(id2(mes10)),nonce(ci2b(mes10)),key(id1(mes10),id2(mes10),pk10)) .
eq (id2(mes10) = intruder) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 3.2) not c-mes3(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
-- assumptions
eq c-mes3(s,mes10,pk10) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep050(e32) .
```

```
close


--> ********************************************************************************** --
--> 4) mes4(s,q10,mes10,mes20,mes30) --
-- 4.1) c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep050(e32) .
close


-- 4.2) not c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
-- assumptions
eq c-mes4(s,q10,mes10,mes20,mes30) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 5) fkm11(s,p10,q10,m10,e10) --
-- 5.1) c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
--eq c-fkm11(s,p10,q10,m10,e10) = true .
eq e10 \in cci1(nw(s)) = true .
-- successor state
```

```
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 5.2) not c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
eq c-fkm11(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 6) fkm12(s,p10,q10,m10,n10,k10) --
-- 6.1) c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm12(s,p10,q10,m10,n10,k10) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder  .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 6.2) not c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm12(s,p10,q10,m10,n10,k10) = false .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 7) fkm21(s,p10,q10,m10,e10,e20) --
-- 7.1) c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
ops e10 e20 : -> Cipher1 .
-- assumptions
-- eq c-fkm21(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq e20 \in cci2b(nw(s)) = true .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep050(e32) .
close


-- 7.2) not c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
eq c-fkm21(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 8) fkm22(s,p10,q10,m10,e10,n20,k20) --
-- 8.1) c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
-- eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep050(e32) .
close


-- 8.2) not c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = false .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep050(e32) .
close
```

```
--> ********************************************************************************** --
--> 9) fkm23(s,p10,q10,m10,e20,n10,k10) --
-- 9.1) c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = true .
eq e20 \in cci2b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 9.2) not c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = false .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 10) fkm24(s,p10,q10,m10,n10,n20,k10,k20) --
-- 10.1) c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
-- eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep050(e32) .
close


-- 10.2) not c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
```

```
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = false .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep050(e32) .
close



--> ******************************************************************************** --
--> 11) fkm31(s,p10,q10,m10,e10,e20) --
-- 11.1) c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq (e32 = e20) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep050(e32) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
--
eq e32 = e20 .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep050(e32) .
close


-- 11.2) not c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
eq c-fkm31(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep050(e32) .
```

```
close


--> ******************************************************************************** --
--> 12) fkm32(s,p10,q10,m10,e10,n20,k20,k30) --
-- 12.1) c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep050(e32) .
close


-- 12.2) not c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep050(e32) .
close



--> ******************************************************************************** --
--> 13) fkm33(s,p10,q10,m10,e20,n10,k10,k30) --
-- 13.1) c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq (e32 = e20) = false .
```

```
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep050(e32) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
--
eq e32 = e20 .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep050(e32) .
close


-- 13.2) not c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 14) fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) --
-- 14.1) c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
```

```
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep050(e32) .
close


-- 14.2) not c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = false .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 15) fkm41(s,p10,q10,m10,e10) --
-- 15.1) c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
eq e10 \in cci4(nw(s)) = true .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep050(e32) .
close


-- 15.2) not c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
eq c-fkm41(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep050(e32) .
close



--> ********************************************************************************** --
--> 16) fkm42(s,p10,q10,m10,n10,k10,k30) --
-- 16.1) c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep050(e32) .
close


-- 16.2) not c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = false .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep050(e32) .
close
```

付 録F　補題3の証明譜

```
--> ********************************************************************************** --
-- I) Base case

open INV
red inv060(init,e4) .
close



-- II) Inductive case
--> ********************************************************************************** --
--> 1) mes1(s,p10,q10,mv10,r10) --
-- 1.1) c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
-- eq c-mes1(s,p10,q10,mv10,r10) = true .
eq mv10 \in um(s) = false .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 1.2) not c-mes1(s,p10,q10,mv10,r10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op mv10 : -> Mvalue .
op r10 : -> Nvalue .
-- assumptions
eq c-mes1(s,p10,q10,mv10,r10) = false .
-- successor state
eq s' = mes1(s,p10,q10,mv10,r10) .
-- check if the predivate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 2) mes2(s,q10,r10,mes10) --
-- 2.1) c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary object
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
op nw10 : -> Network .
```

```
-- assumptions
-- eq c-mes2(s,q10,r10,mes10) = true .
eq nw(s) = mes10 , nw10 .
eq m1?(mes10) = true .
eq reciver1(mes10) = q10 .
eq r10 \in uv(s) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 2.2) not c-mes2(s,q10,r10,mes10)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
op r10 : -> Nvalue .
op mes10 : -> Msg .
-- assumptions
 eq c-mes2(s,q10,r10,mes10) = false .
-- successor state
eq s' = mes2(s,q10,r10,mes10) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 3) mes3(s,mes10,pk10) --
-- 3.1) c-mes(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
op nw10 : -> Network .
-- assumptions
-- eq c-mes3(s,mes10,pk10) = true .
eq nw(s) = mes10 , nw10 .
eq m2?(mes10) = true .
eq reciver2(mes10) = ca .
eq pr0(skey(ci2a(mes10))) = id1(mes10) .
eq pr0(skey(ci2b(mes10))) = id2(mes10) .
eq mnonce(mes10) = mnonce(ci2a(mes10)) .
eq mnonce(mes10) = mnonce(ci2b(mes10)) .
eq mnonce(ci2a(mes10)) = mnonce(ci2b(mes10)) .
eq p(ci2a(mes10)) = id1(mes10) .
eq q(ci2a(mes10)) = id2(mes10) .
eq p(ci2b(mes10)) = id1(mes10) .
eq q(ci2b(mes10)) = id2(mes10) .
eq pk10 \in sk(s) = false .
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 3.2) not c-mes3(s,mes10,pk10)
open ISTEP
-- arbitrary objects
op mes10 : -> Msg .
op pk10 : -> Pubkey .
-- assumptions
eq c-mes3(s,mes10,pk10) = false .
```

```
-- successor state
eq s' = mes3(s,mes10,pk10) .
-- check if the predicate is true
red istep060(e4) .
close



--> ******************************************************************************** --
--> 4) mes4(s,q10,mes10,mes20,mes30) --
-- 4.1) c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq (e4 = ci3a(mes30)) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep060(e4) .
close


open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
```

```
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq e4 = ci3a(mes30) .
eq pr0(skey(ci3a(mes30))) = intruder .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep060(e4) .
close


open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
op nw10 : -> Network .
-- assumptions
-- eq c-mes4(s,q10,mes10,mes20,mes30) = true .
eq nw(s) = mes10 , mes20 , mes30 , nw10 .
eq m1?(mes10) = true .
eq m2?(mes20) = true .
eq m3?(mes30) = true .
eq reciver1(mes10) = q10 .
eq sender1(mes20) = q10 .
eq reciver2(mes20) = ca .
eq pr0(skey(ci2b(mes20))) = q10 .
eq mnonce(mes10) = mnonce(mes20) .
eq mnonce(mes20) = mnonce(ci2b(mes20)) .
eq id2(mes20) = q10 .
eq p(ci2b(mes20)) = id1(mes20) .
eq q(ci2b(mes20)) = id2(mes20) .
eq sender2(mes30) = ca .
eq reciver1(mes30) = q10 .
eq pr0(skey(ci3b(mes30))) = q10 .
eq mnonce(mes30) = mnonce(mes20) .
eq nonce(ci3b(mes30)) = nonce(ci2b(mes20)) .
--
eq e4 = ci3a(mes30) .
eq (pr0(skey(ci3a(mes30))) = intruder) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep060(e4) .
close


-- 4.2) not c-mes4(s,q10,mes10,mes20,mes30)
open ISTEP
-- arbitrary objects
op q10 : -> Principal .
ops mes10 mes20 mes30 : -> Msg .
-- assumptions
eq c-mes4(s,q10,mes10,mes20,mes30) = false .
-- successor state
eq s' = mes4(s,q10,mes10,mes20,mes30) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************* --
--> 5) fkm11(s,p10,q10,m10,e10) --
```

```
-- 5.1) c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
--eq c-fkm11(s,p10,q10,m10,e10) = true .
eq e10 \in cci1(nw(s)) = true .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 5.2) not c-fkm11(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
-- assumptions
eq c-fkm11(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm11(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 6) fkm12(s,p10,q10,m10,n10,k10) --
-- 6.1) c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm12(s,p10,q10,m10,n10,k10) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder  .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 6.2) not c-fkm12(s,p10,q10,m10,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm12(s,p10,q10,m10,n10,k10) = false .
-- successor state
eq s' = fkm12(s,p10,q10,m10,n10,k10) .
-- check if the predicate is true
red istep060(e4) .
close
```

```
--> ******************************************************************************** --
--> 7) fkm21(s,p10,q10,m10,e10,e20) --
-- 7.1) c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
-- eq c-fkm21(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq e20 \in cci2b(nw(s)) = true .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep060(e4) .
close


-- 7.2) not c-fkm21(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher1 .
-- assumptions
eq c-fkm21(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm21(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep060(e4) .
close



--> ******************************************************************************** --
--> 8) fkm22(s,p10,q10,m10,e10,n20,k20) --
-- 8.1) c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
-- eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = true .
eq e10 \in cci2a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep060(e4) .
close


-- 8.2) not c-fkm22(s,p10,q10,m10,e10,n20,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher1 .
```

```
op n20 : -> Nonce .
op k20 : -> Skey .
-- assumptions
eq c-fkm22(s,p10,q10,m10,e10,n20,k20) = false .
-- successor state
eq s' = fkm22(s,p10,q10,m10,e10,n20,k20) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************* --
--> 9) fkm23(s,p10,q10,m10,e20,n10,k10) --
-- 9.1) c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
-- eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = true .
eq e20 \in cci2b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 9.2) not c-fkm23(s,p10,q10,m10,e20,n10,k10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher1 .
op n10 : -> Nonce .
op k10 : -> Skey .
-- assumptions
eq c-fkm23(s,p10,q10,m10,e20,n10,k10) = false .
-- successor state
eq s' = fkm23(s,p10,q10,m10,e20,n10,k10) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************* --
--> 10) fkm24(s,p10,q10,m10,n10,n20,k10,k20) --
-- 10.1) c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
-- eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
```

```
eq pr0(k20) = intruder .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep060(e4) .
close


-- 10.2) not c-fkm24(s,p10,q10,m10,n10,n20,k10,k20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
-- assumptions
eq c-fkm24(s,p10,q10,m10,n10,n20,k10,k20) = false .
-- successor state
eq s' = fkm24(s,p10,q10,m10,n10,n20,k10,k20) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 11) fkm31(s,p10,q10,m10,e10,e20) --
-- 11.1) c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
-- eq c-fkm31(s,p10,q10,m10,e10,e20) = true .
eq e10 \in cci3a(nw(s)) = true .
eq e20 \in cci3b(nw(s)) = true .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep060(e4) .
close


-- 11.2) not c-fkm31(s,p10,q10,m10,e10,e20)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops e10 e20 : -> Cipher2 .
-- assumptions
eq c-fkm31(s,p10,q10,m10,e10,e20) = false .
-- successor state
eq s' = fkm31(s,p10,q10,m10,e10,e20) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 12) fkm32(s,p10,q10,m10,e10,n20,k20,k30) --
-- 12.1) c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
```

```
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = true .
eq e10 \in cci3a(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep060(e4) .
close


-- 12.2) not c-fkm32(s,p10,q10,m10,e10,n20,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
op n20 : -> Nonce .
op k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm32(s,p10,q10,m10,e10,n20,k20,k30) = false .
-- successor state
eq s' = fkm32(s,p10,q10,m10,e10,n20,k20,k30) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 13) fkm33(s,p10,q10,m10,e20,n10,k10,k30) --
-- 13.1) c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = true .
eq e20 \in cci3b(nw(s)) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep060(e4) .
close


-- 13.2) not c-fkm33(s,p10,q10,m10,e20,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
```

```
op e20 : -> Cipher2 .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm33(s,p10,q10,m10,e20,n10,k10,k30) = false .
-- successor state
eq s' = fkm33(s,p10,q10,m10,e20,n10,k10,k30) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 14) fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) --
-- 14.1) c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq n20 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq pr0(k20) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep060(e4) .
close


-- 14.2) not c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
ops n10 n20 : -> Nonce .
ops k10 k20 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) = false .
-- successor state
eq s' = fkm34(s,p10,q10,m10,n10,n20,k10,k20,k30) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 15) fkm41(s,p10,q10,m10,e10) --
-- 15.1) c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
```

```
eq e10 \in cci4(nw(s)) = true .
--
eq (e4 = e10) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep060(e4) .
close


open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
-- eq c-fkm41(s,p10,q10,m10,e10) = true .
eq e10 \in cci4(nw(s)) = true .
--
eq e4 = e10 .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep060(e4) .
close


-- 15.2) not c-fkm41(s,p10,q10,m10,e10)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op e10 : -> Cipher2 .
-- assumptions
eq c-fkm41(s,p10,q10,m10,e10) = false .
-- successor state
eq s' = fkm41(s,p10,q10,m10,e10) .
-- check if the predicate is true
red istep060(e4) .
close



--> ********************************************************************************** --
--> 16) fkm42(s,p10,q10,m10,n10,k10,k30) --
-- 16.1) c-fkm42(s,p10,q10,m10,n10,k10,k30)
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
-- eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = true .
eq n10 \in cnonce(nw(s)) = true .
eq pr0(k10) = intruder .
eq k30 \in ckey(nw(s)) = true .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep060(e4) .
close


-- 16.2) not c-fkm42(s,p10,q10,m10,n10,k10,k30)
```

```
open ISTEP
-- arbitrary objects
ops p10 q10 : -> Principal .
op m10 : -> Mnonce .
op n10 : -> Nonce .
op k10 : -> Skey .
op k30 : -> Key .
-- assumptions
eq c-fkm42(s,p10,q10,m10,n10,k10,k30) = false .
-- successor state
eq s' = fkm42(s,p10,q10,m10,n10,k10,k30) .
-- check if the predicate is true
red istep060(e4) .
close
```