



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)  
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش پروژه درس پروتکل‌های امنیتی

عنوان

**تحلیل صوری پروتکل Kerberos نسخه ۵**

نگارش

احسان عدالت ۹۴۱۳۱۰۹۰  
سید محمد مهدی احمدپناه ۹۴۱۳۱۰۸۶

استاد درس

دکتر بابک صادقیان

تابستان ۱۳۹۵

## فهرست مطالب

۱	مقدمه.....	۱
۲	پروتکل Kerberos نسخه ۵.....	۳
۲,۱	شرح پروتکل.....	۳
۲,۲	حملات روی پروتکل Kerberos.....	۵
۳	ابزارهای تحلیل پروتکل‌های امنیتی.....	۶
۳,۱	ابزار PRISM.....	۶
۳,۲	ابزار Isabelle.....	۱۱
۳,۳	ابزار Maude-NPA.....	۱۲
۴	زبان Maude-PSL.....	۱۶
۴,۱	قسمت Theory.....	۱۶
۴,۲	قسمت Protocol.....	۱۸
4.3	قسمت Intruder.....	۱۹
۴,۴	قسمت Attacks.....	۲۰
۴,۵	درستی‌سنجی پروتکل Kerberos نسخه ۵ با استفاده از Maude-PSL.....	۲۱
۴,۶	درستی‌سنجی پروتکل Kerberos نسخه ۱ با استفاده از Maude-PSL.....	۲۶
۵	مراجع.....	۳۱
۶	پیوست‌ها.....	۳۲
۶,۱	کد تحلیل پروتکل Kerberos نسخه ۵ با ابزار Isabelle.....	۳۲
۶,۲	کد تحلیل پروتکل Needham-Schröder با ابزار Maude-NPA.....	۵۱
۶,۳	نمونه حمله یافت شده توسط ابزار Maude-NPA.....	۵۳

۵۴.....	۶,۴ پروتکل Needham-Schröder به زبان Maude-PSL
---------	---

## فهرست شکل‌ها

- شکل ۱: مراحل پروتکل Kerberos نسخه ۵ ..... ۴
- شکل ۲: نمایی از ابزار PRISM و بررسی خاصیت درستی در مسئله شام خوردن رمازها ..... ۱۰
- شکل ۳: نمایی از ابزار PRISM و توصیف مسئله شام خوردن رمازها ..... ۹
- شکل ۴: نمایی از ابزار PRISM و بررسی خاصیت بی‌نامی در مسئله رمازها ..... ۱۰
- شکل ۵: نمایی از ابزار Isabelle ..... ۱۱
- شکل ۶: نمایی از ابزار Maude-NPA و بارگذاری کد پروتکل Needham-Schrueder در آن و اجرای دستور  
. red genGrammars ..... ۱۲
- شکل ۷: اجرای دستور red summary و red initials و دریافت کد حمله ..... ۱۳
- شکل ۸: نمایی از ابزار Maude-NPA در سیستم عامل OSX ..... ۱۴
- شکل ۹: اجرای پروتکل دفی-هلمن در ابزار Maude-NPA در سیستم عامل OSX و بروز خطا در آن ..... ۱۵
- شکل ۱۰: پروتکل دفی-هلمن ..... ۱۸
- شکل ۱۱: تبدیل زبان ps1 به npa برای پروتکل Needham-schrueder ..... ۲۱

## ۱ مقدمه

پروتکل‌های امنیتی از جمله مهمترین ابزارهای برقراری ارتباط امن بین افراد است. همان طور که در درس هم اشاره شد، این پروتکل‌ها در کاربردهای مختلف از جمله تصدیق اصالت طرفین، امضا، توزیع کلید و غیره مورد استفاده قرار می‌گیرد. پروتکل‌هایی که طراحی می‌شوند ممکن است دارای نواقصی باشند که مهاجم می‌تواند از آنها سوء استفاده کرده و اطلاعات محرمانه افراد را شنود کند. یکی از راه‌های بررسی و درستی‌سنجی پروتکل‌ها، بیان صوری آنها و سپس استفاده از ابزارهای راستی‌آزمایی در این حوزه است. ابزارهای مختلف با رویکرد و کاربردهای مختلف و سطح توانایی گوناگون در توصیف و درستی‌سنجی پروتکل‌ها تا به حال ارائه شده‌اند.

ابزارهایی که در حوزه درستی‌سنجی پروتکل‌ها استفاده شده است را می‌توان در سه دسته کلی دسته‌بندی کرد. دسته اول ابزارهای دستیار اثبات<sup>۱</sup> مانند ابزار Isabelle [۱] هستند. این نوع ابزارها عام‌منظوره هستند و برای توصیف ریاضی هر سیستمی و درستی‌سنجی آنها استفاده می‌شود. کارکردن با این نوع از ابزارها برای تحلیل پروتکل‌های امنیتی دشوار است، چون نیاز به پیشینه ریاضی قوی برای تحلیل دارد. دسته دوم ابزارهای بررسی‌کننده مدل<sup>۲</sup> مانند ابزار PRISM [۲] است. این نوع ابزارها نیز عام‌منظوره هستند و برای تحلیل هر سیستمی استفاده می‌شوند. کار کردن با این دسته از ابزارها هم دشوار است، چون در ابتدا لازم است مفاهیم اولیه مورد نیاز در توصیف پروتکل‌های امنیتی مثل نانس، مهر زمانی و غیره را در آنها تعریف کرد و سپس کار تحلیل پروتکل را شروع کرد. دسته سوم، ابزارهای خاص‌منظوره تحلیل پروتکل‌های امنیتی مثل ProVerif، Scyther و Maude-PSL [۳] هستند. کار کردن با این نوع از ابزارها عموماً آسان‌تر است، چون رویکرد این ابزارها تحلیل پروتکل‌های امنیتی بوده و بعضی از مفاهیم مورد نیاز برای توصیف و تحلیل آنها در این دسته از ابزارها از پیش تعریف شده است.

در این پروژه هدف تحلیل پروتکل Kerberos نسخه ۵ است که در فصل ۲ به تفصیل توضیح داده خواهد شد. برای تحلیل این پروتکل ابزارهای مختلفی مورد بررسی قرار گرفت. ابزارهای Isabelle، PRISM، Real-Time Maude [۴] و Maude-NPA [۳] مورد بررسی قرار گرفتند که در فصل ۳ ویژگی‌های هر یک بیان خواهد شد. در نهایت ابزار Maude-PSL برای تحلیل پروتکل انتخاب شد. ویژگی‌های این ابزار و نحوه توصیف پروتکل

<sup>۱</sup> Proof Assistant

<sup>۲</sup> Model Checker

در آن در فصل ۴ بیان خواهد شد. در نهایت، توصیف پروتکل Kerberos نسخه ۵ با استفاده از این ابزار آمده است.

## ۲ پروتکل Kerberos نسخه ۵

پروتکل Kerberos یک پروتکل تصدیق هویت است که توسط دانشگاه MIT برای تصدیق هویت کارساز<sup>۳</sup> و کارخواه<sup>۴</sup> در یک شبکه ارائه شده است. این پروتکل به افراد حاضر در آن این امکان را می‌دهد تا بدون افشای مقداری سری هویت خود را اثبات کنند. این پروتکل از رمزنگاری متقارن استفاده می‌کند و نیاز به فرد ثالث مورد اعتماد دارد. نسخه اول این پروتکل برای اولین بار در دهه ۱۹۸۰ در دانشگاه MIT مورد استفاده قرار گرفت. آخرین نسخه این پروتکل در سال ۲۰۰۵ توسط MIT ارائه شده است و تحقیقات بر روی آن هنوز در جریان است. شرکت مایکروسافت از این پروتکل به عنوان روش پیش‌فرض برای تصدیق اصالت در سیستم‌های خود استفاده می‌کند.

### ۲.۱ شرح پروتکل

در دنیای واقعی وقتی فردی بخواهد هویت خود را به دیگری اثبات کند از کارت شناسایی یا گواهی‌نامه رانندگی استفاده می‌کند. در پروتکل Kerberos هم کارخواه برای اثبات هویت خود از بلیت<sup>۵</sup> استفاده می‌کند. این بلیت همانند کارت شناسایی که شامل نام، آدرس و تاریخ تولد است، شامل اطلاعاتی در مورد کارخواه مثل آدرس فیزیکی (آدرس IP) آن است. همان طور که مشخص است صرف وجود کارت شناسایی برای تصدیق هویت کافی نیست و باید عکس موجود روی آن با فرد مطابقت داده شود. در پروتکل هم در کنار بلیت یک تصدیق اصالت‌کننده<sup>۶</sup> قرار می‌گیرد. به مجموعه بلیت و تصدیق اصالت‌کننده اعتبارنامه<sup>۷</sup> گفته می‌شود. همانند کارت شناسایی که باید از یک نهاد معتبر و مورد اعتماد مثل ثبت احوال دریافت شود، در این پروتکل هم اعتبار نامه از فرد ثالث مورد اعتماد توزیع‌کننده کلید<sup>۸</sup> دریافت می‌شود که در اینجا به دو قسمت سرویس تصدیق اصالت‌کننده Kerberos (KAS) و سرویس اعطای بلیت (TGS) تقسیم می‌شود. در این پروتکل سه مرحله وجود دارد؛ مرحله لاگین<sup>۹</sup> که فرد اعتبارنامه لازم را از KDC دریافت می‌کند. مرحله درخواست سرویس و مرحله

<sup>۳</sup> Server

<sup>۴</sup> Client

<sup>۵</sup> Ticket

<sup>۶</sup> Authenticator

<sup>۷</sup> Credential

<sup>۸</sup> Key Distribution Center (KDC)

<sup>۹</sup> Login

استفاده از سرویس. برای توضیح گذرهای مختلف پروتکل از مثال دریافت بلیت هواپیما توسط آلیس استفاده خواهد شد.

ابتدا آلیس برای دریافت بلیط استفاده از یک سرویس خاص به KDC مراجعه می‌کند. KDC یک کلید نشست برای ارتباط بین آلیس و سرویس را همراه با نام آلیس و مدت اعتبار آن، به وسیله کلید آن سرویس که فقط در اختیار آن سرویس است رمز می‌کند.

$$Ticket: \{Timestamp, Alice, K_{Alice,service}, Lifetime\}_{K_{Service}}$$

این بلیت به همراه یک رسید که شامل  $K_{Alice,service}$  است در اختیار آلیس قرار می‌گیرد. آلیس هرگاه بخواهد از سرویس استفاده کند بلیط را به سرویس ارائه می‌دهد. سرویس مطمئن است که هیچ کس از محتوای بلیط مطلع نیست چون با کلیدی که بین خودش و KDC مشترک بوده است رمز شده است. همچنین وجود نام آلیس در بلیط نشان‌دهنده این است که سرویس در حال تعامل با آلیس است. در نهایت با استفاده از تصدیق اصالت‌کننده که تنها شامل مهرزمانی است ولی فقط آلیس و سرویس می‌توانند از محتوای آن مطلع شوند، آلیس نشان می‌دهد که کلید  $K_{Alice,service}$  را در اختیار دارد.

$$Authenticator: \{Timestamp\}_{K_{Alice,Service}}$$

در این پروتکل علاوه بر بلیت بیان شده در بالا، بلیت دیگری بین KAS و TGS وجود دارد که آلیس از آن برای ارتباط با TGS و دریافت بلیت ارتباط با سرویس استفاده می‌کند. زمان اعتبار این بلیت مثل زمان اعتبار کارت شناسایی، نسبت به بلیت ارتباط با سرویس بیشتر است. در ادامه مراحل پروتکل آورده شده است:

1.  $Alice \xrightarrow{TGS, Alice, Nonce} KAS$
2.  $Alice \xleftarrow{\{K_{Alice,TGS}, Nonce, TGS\}_{K_{Alice}}, \{TGT\}_{K_{TGS}}} KAS$
3.  $Alice \xrightarrow{\{Authenticator\}_{K_{Alice,TGS}}, \{TGT\}_{K_{TGS}}, Airline, Alice, Nonce'} TGS$
4.  $Alice \xleftarrow{\{K_{Alice,Airline}, Nonce', Airline\}_{K_{Alice,TGS}}, \{Ticket\}_{K_{Airline}}} TGS$
5.  $Alice \xrightarrow{\{Authenticator\}_{K_{Alice,Airline}}, \{Ticket\}_{K_{Airline}}} Airline$
- (6.)  $Alice \xleftarrow{\{T\}_{K_{Alice,Airline}}} Airline$

شکل ۱: مراحل پروتکل Kerberos نسخه ۵ [۵]



گذرهای اول و دوم پروتکل به منظور لاگین شده در سیستم اجرا می‌شود. بعد از گذر دوم آلیس کلید نشست ارتباط با TGS را که KAS تولید کرده است را دریافت می‌کند. اکنون آلیس از بلیت TGT برای اثبات هویت خود به TGS البته قبل از زمان انقضای آن می‌تواند استفاده کند.

گذرهای سوم و چهارم پروتکل برای درخواست استفاده از سرویس استفاده می‌شود. آلیس از بلیت TGT برای اثبات هویت خود به TGS استفاده می‌کند. در نهایت TGS بلیت Ticket را برای ارتباط با سرویس هواپیمایی، در اختیار آلیس قرار می‌دهد و آلیس می‌تواند هویت خود را با آن به سرویس اثبات کند.

گذر پنجم هم برای درخواست استفاده از سرویس استفاده می‌شود. آلیس بلیط خود را همراه با تصدیق اصالت‌کننده به سرویس ارائه می‌کند. در صورت صحت بلیت و اعتبار زمانی آن، آلیس می‌تواند از سرویس استفاده کند.ش

گذر نهایی پروتکل اختیاری است. در این گذر سرویس مهرزمانی دریافتی در گذر قبلی را با یک اضافه کرده و به آلیس بر می‌گرداند. این گذر به منظور اجرای تصدیق اصالت دو طرفه به صورت اختیاری قابل اجرا است.

توضیحاتی که در مورد پروتکل Kerberos نسخه ۵ در این قسمت بیان شد از پایان‌نامه مارتین گریملند [۵] در سال ۲۰۰۶ استفاده شده است. در این پایان‌نامه این پروتکل با استفاده از زبان بلادرنگ Real-Time-Maude تحلیل شده است.

## ۲,۲ حملات روی پروتکل Kerberos

همان‌طور که در [۷] آمده است، حملات زیر روی پروتکل Kerberos قابل انجام است.

- Replay Attack
- Secure Time Service Attack
- Password Guessing Attack
- Spoofing Login Attack
- Inter-Session Chosen PlainText Attack
- Exposure of Session Key Attack
- The Scope of Tickets Attack

### ۳ ابزارهای تحلیل پروتکل‌های امنیتی

در این قسمت ابزارهایی که برای تحلیل پروتکل Kerberos بررسی شدند به طور مختصر بیان می‌شوند و ویژگی‌های هر یک و تحلیل پروتکلی ساده به وسیله آنها بیان می‌شود و در نهایت دلیل اینکه هر کدام از آنها برای تحلیل Kerberos استفاده نشدند بیان می‌شود.

#### ۳.۱ ابزار PRISM

PRISM یک بررسی‌کننده مدل احتمالاتی برای تحلیل صوری سیستم‌هایی است که رفتار دلخواه<sup>۱۰</sup> یا احتمالاتی دارند. PRISM یک ابزار عام منظوره است که از آن برای تحلیل پروتکل‌های امنیتی و ارتباطاتی، الگوریتم‌های دلخواه توزیع‌شده، سیستم‌های زیستی و بسیاری دیگر از سیستم‌ها استفاده شده است.

PRISM می‌تواند مدل‌های احتمالاتی بسیاری را بسازد و تحلیل کند از جمله:

- زنجیره مارکوف در زمان گسسته (DTMC)
- زنجیره مارکوف در زمان پیوسته (CTMC)
- پردازش‌های تصمیم‌گیری مارکوف (MDP)
- آتاماتای احتمالاتی
- آتاماتای احتمالاتی زمان‌دار

PRISM از یک زبان مبتنی بر حالت برای توصیف سیستم‌های مختلف استفاده می‌کند و توانایی پاسخ‌گویی به این نوع از سوالات در مورد سیستم‌های مختلف را دارد: «احتمال این که یک پروتکل امنیتی از حالت اولیه به خطا برسد چقدر است؟» یا «اندازه مورد انتظار صف بعد از گذشت ۳۰ دقیقه چقدر است؟».

در ادامه توصیف مسئله شام خوردن رمازها (برای سادگی برای ۳ نفر) با استفاده از این ابزار بیان شده و نحوه اجرا و ارزیابی آن توضیح داده می‌شود.

مدل این مسئله به زبان PRISM همراه با توضیحات آن در ادامه آمده است:

```

// model of dining cryptographers
// gxn/dxp 15/11/06

mdp

// number of cryptographers

const int N = 3;

// constants used in renaming (identities of cryptographers)

const int p1 = 1;
const int p2 = 2;
const int p3 = 3;

// global variable which decides who pays
// (0 - master pays, i=1..N - cryptographer i pays)
global pay : [0..N];

// module for first cryptographer
module crypt1
    coin1 : [0..2]; // value of its coin
    s1 : [0..1]; // its status (0 = not done, 1 = done)
    agree1 : [0..1]; // what it states (0 = disagree, 1 = agree)
    // flip coin
    [] coin1=0 -> 0.5 : (coin1'=1) + 0.5 : (coin1'=2);
    // make statement (once relevant coins have been flipped)
    // agree (coins the same and does not pay)
    [] s1=0 & coin1>0 & coin2>0 & coin1=coin2 & (pay!=p1) -> (s1'=1) &
    (agree1'=1);
    // disagree (coins different and does not pay)
    [] s1=0 & coin1>0 & coin2>0 & !(coin1=coin2) & (pay!=p1) -> (s1'=1);
    // disagree (coins the same and pays)
    [] s1=0 & coin1>0 & coin2>0 & coin1=coin2 & (pay=p1) -> (s1'=1);
    // agree (coins different and pays)
    [] s1=0 & coin1>0 & coin2>0 & !(coin1=coin2) & (pay=p1) -> (s1'=1) &
    (agree1'=1);
    // synchronising loop when finished to avoid deadlock
    [done] s1=1 -> true
endmodule

// construct further cryptographers with renaming

module crypt2 = crypt1 [ coin1=coin2, s1=s2, agree1=agree2, p1=p2, coin2=coin3 ]
endmodule

module crypt3 = crypt1 [ coin1=coin3, s1=s3, agree1=agree3, p1=p3, coin2=coin1 ]
endmodule

```

```
// set of initial states
// (cryptographers in their initial state, "pay" can be anything)
init coin1=0&s1=0&agree1=0 & coin2=0&s2=0&agree2=0 & coin3=0&s3=0&agree3=0
endinit

// unique integer representing outcome
formula outcome = 4*agree1 + 2*agree2 + 1*agree3 ;
// parity of number of "agree"s (0 = even, 1 = odd)
formula parity = func(mod, agree1+agree2+agree3, 2);
// label denoting states where protocol has finished
label "done" = s1=1&s2=1&s3=1;
// label denoting states where number of "agree"s is even
label "even" = func(mod, (agree1+agree2+agree3), 2)=0;
// label denoting states where number of "agree"s is even
label "odd" = func(mod, (agree1+agree2+agree3), 2)=1;
```

مدل بالا از دو جنبه درستی<sup>۱۱</sup> و بی‌نامی<sup>۱۲</sup> مورد بررسی قرار می‌گیرد که کد آنها در ادامه آمده است:

درستی:

```
// Correctness for the case where the master pays
// (final parity of number of number of "agrees"s matches that of N)
(pay=0) => P>=1 [ F "done" & parity=func(mod, N, 2) ]
// Correctness for the case where a cryptographer pays
// (final parity of number of number of "agrees"s does not match that of N)
(pay>0) => P>=1 [ F "done" & parity!=func(mod, N, 2) ]
```

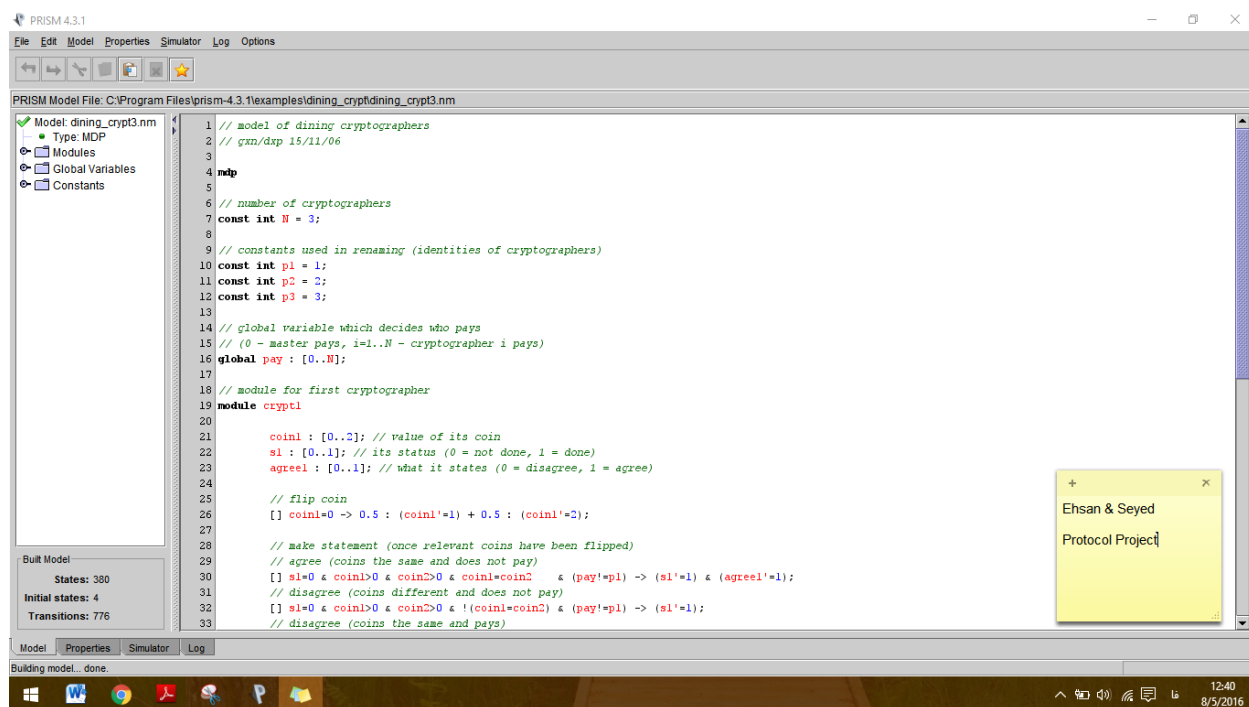
بی‌نامی:

```
const int k;
// Anonymity - check for  $k=0..2^N$  - both min/max should be the same and equal to  $1/2^{(N-1)}$  or 0
// (depending on the parity of the number of bits in the binary representation of outcome)
Pmin=? [ F "done" & outcome = k {"init"&pay>0}{min} ]
```

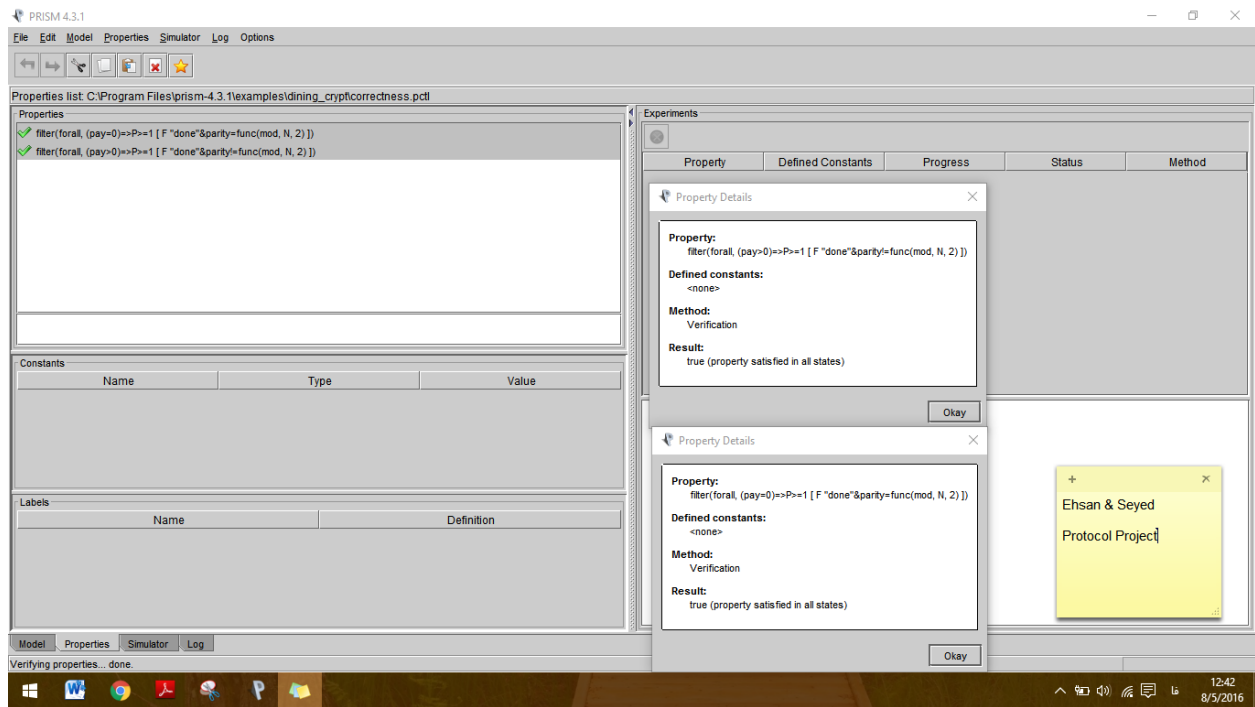
<sup>۱۱</sup> Correctness

<sup>۱۲</sup> Anonymity

$P_{max}=?$  [ F "done" & outcome = k { "init"&pay>0 } {max} ]

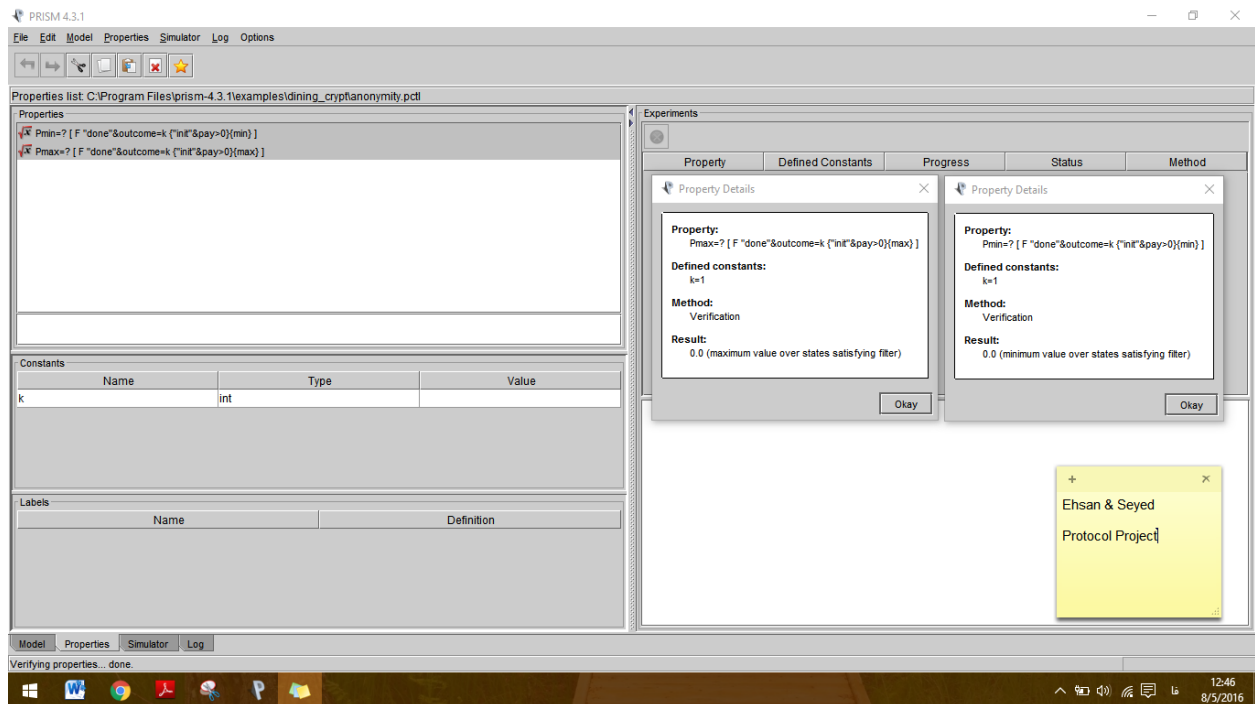


شکل ۲: نمایی از ابزار PRISM و توصیف مسئله شام خوردن رمازها



شکل ۴: نمایی از ابزار PRISM و بررسی خاصیت درستی در مسئله شام خوردن رمازها

همان طور که در توصیف مسئله شام خوردن رمازها و بررسی خاصیت‌های درستی و بی‌نامی مشخص است،



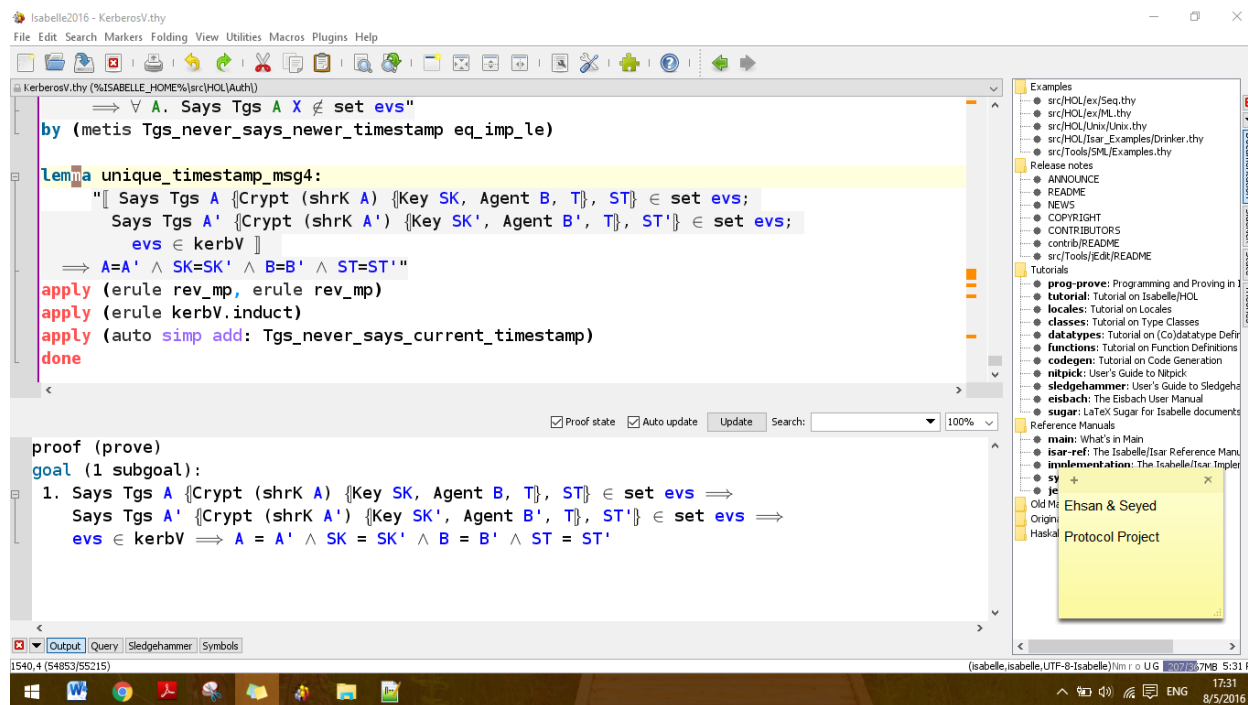
شکل ۳: نمایی از ابزار PRISM و بررسی خاصیت بی‌نامی در مسئله رمازها

استفاده از این ابزار در توصیف پروتکل‌های امنیتی بسیار دشوار خواهد بود چون که لازم است ابتدا توصیف پروتکل به صورت صوری بیان شود سپس در مورد حالات مختلف آن از نظر احتمالاتی فکر شود تا بتوان توصیف آن پروتکل را به زبان PRISM بیان کرد. این مطلب در مقاله [۶] آمده است. در این مقاله پروتکل‌های Needham-Schroeder و TMN به زبان PRISM توصیف شده‌اند برای این منظور ابتدا این پروتکل‌ها به زبان ابزار ProVerif که ابزار توصیف پروتکل‌های امنیتی است توصیف شده سپس به زبان PRISM تبدیل شده است. این موضوع دشواری توصیف به زبان PRISM را به روشنی نشان می‌دهد.

## ۳.۲ ابزار Isabelle

Isabelle یک دستیار اثبات عام منظوره است. با این ابزار می‌توان فرمول‌های ریاضی را به شکل صوری بیان کرد و سپس توسط این ابزار و محاسبات منطقی این فرمول‌ها را اثبات کرد. کاربرد اصلی این ابزار بیان صوری اثبات‌های ریاضی و درستی‌سنجی صوری است که اثبات درستی سخت‌افزار یا نرم‌افزارهای کامپیوتری یا خاصیت‌های زبان‌های کامپیوتری و یا پروتکل‌ها با آن انجام می‌شود.

در ادامه تصویری از محیط این ابزار همراه با قسمت نهایی تحلیل پروتکل Kerberos نسخه ۵ آمده است. در قسمت پیوست کد کامل آن در قسمت ۶.۱ قرار داده شده است. همان طور که مشخص است تحلیل پروتکل‌ها به کمک Isabelle کار بسیار دشواری است و تعداد خطوط تحلیل Kerberos نسخه ۵ برابر با ۱۵۵۰



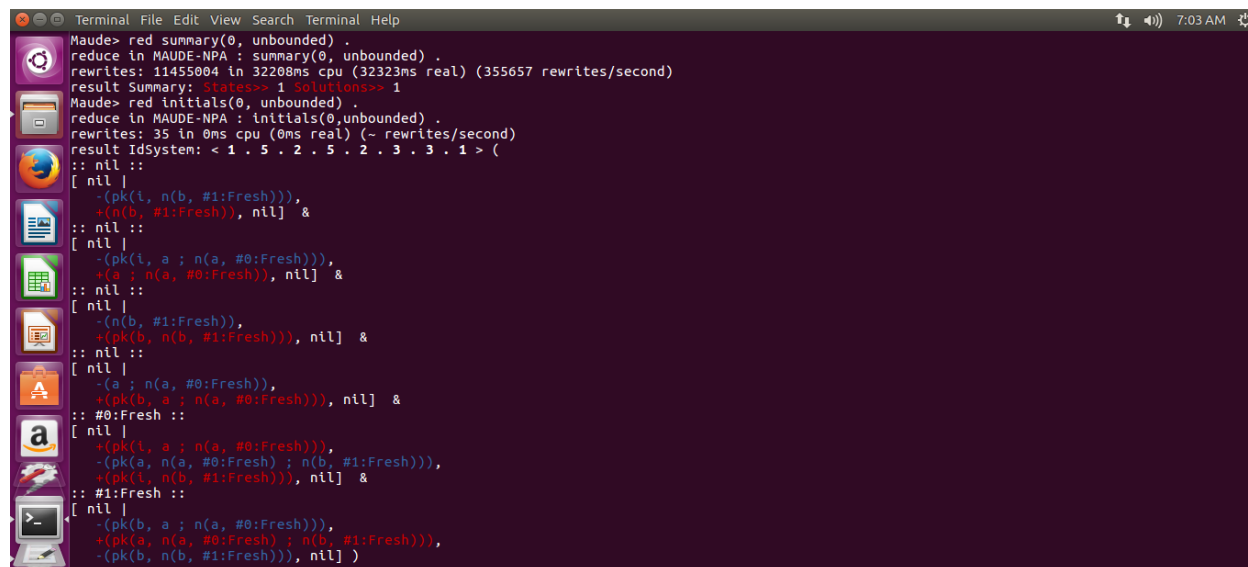
شکل ۵: نمایی از ابزار Isabelle





در شکل ۶ نمایی از ابزار Maude-NPA و بارگذاری کد پروتکل Needham-Schrueder در آن و اجرای دستور red genGrammars آمده است. دستور red genGrammars برای افزایش سرعت در اجرای حمله و یافتن حالات مورد نظر استفاده می‌شود. در شکل ۷ هم دستورات red summary برای یافتن مسیری از حالت حمله به حالت ابتدایی استفاده می‌شود. در صورت یافتن مسیر خروجی آن در برابر solution یک می‌شود. با استفاده از دستور red initials مسیر حمله و کد اجرای حمله نشان داده می‌شود که در پیوست در قسمت ۳، ۶، ۳ آمده است. در کل برای یافتن تمام حملات ممکن باید دستور red summary را برای اعداد ۱، ۲، ۳ ... استفاده کرد. در صورتی که خروجی یک بود red run یا red initials را اجرا کرد تا کد حمله نمایان شود.

همان طور که دیده می‌شود این ابزار برای یافتن تمام حملات ممکن روی یک پروتکل بسیار مفید است و از آنجایی که ویژگی‌ها و خاصیت‌های مختلف پروتکل‌های امنیتی را پشتیبانی می‌کند برای تعریف و تحلیل پروتکل‌ها بسیار مفید است. تنها مشکلی که این ابزار دارد زبان سخت آن در توصیف پروتکل‌های امنیتی است. در سال ۲۰۱۵ ابزار جدید به نام Maude-PSL معرفی شد که در توصیف پروتکل زبانی بسیار ساده‌تر و شبیه به شکل عمومی و آشنای آلیس-باب دارد. علاوه بر آن بر روی Maude-NPA قرار می‌گیرد و تمام ویژگی‌ها و مشخصات آن را نیز پشتیبانی می‌کند. در قسمت ۴ این ابزار معرفی خواهد شد.

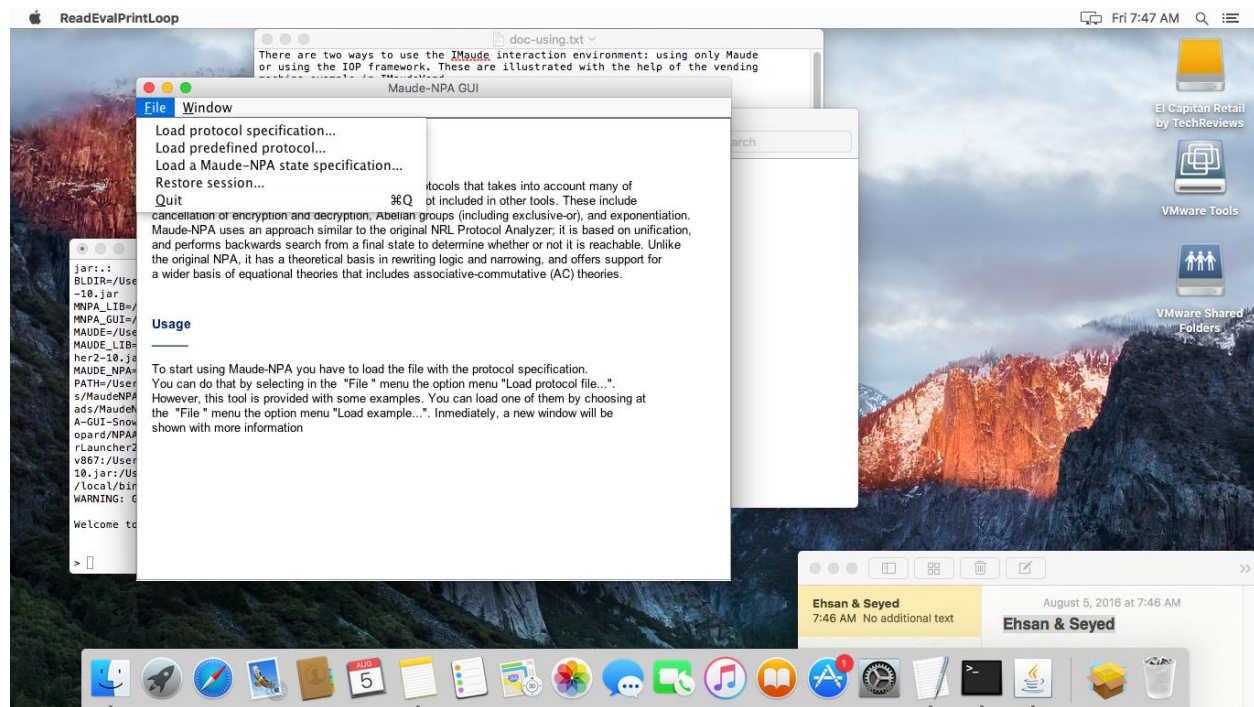


```

Maude> red summary(0, unbounded) .
reduce in MAUDE-NPA : summary(0, unbounded) .
rewrites: 11455004 in 32208ms cpu (32323ms real) (355657 rewrites/second)
result Summary: States>> 1 Solutions>> 1
Maude> red initials(0, unbounded) .
reduce in MAUDE-NPA : initials(0, unbounded) .
rewrites: 35 in 0ms cpu (0ms real) (~ rewrites/second)
result IdSystem: < 1 . 5 . 2 . 5 . 2 . 3 . 3 . 1 > (
:: nil ::
[ nil |
  -(pk(l, n(b, #1:Fresh))),
  +(n(b, #1:Fresh)), nil] &
:: nil ::
[ nil |
  -(pk(l, a ; n(a, #0:Fresh))),
  +(a ; n(a, #0:Fresh)), nil] &
:: nil ::
[ nil |
  -(n(b, #1:Fresh)),
  +(pk(b, n(b, #1:Fresh))), nil] &
:: nil ::
[ nil |
  -(a ; n(a, #0:Fresh)),
  +(pk(b, a ; n(a, #0:Fresh))), nil] &
:: #0:Fresh ::
[ nil |
  +(pk(l, a ; n(a, #0:Fresh))),
  -(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),
  +(pk(l, n(b, #1:Fresh))), nil] &
:: #1:Fresh ::
[ nil |
  -(pk(b, a ; n(a, #0:Fresh))),
  +(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),
  -(pk(b, n(b, #1:Fresh))), nil] )

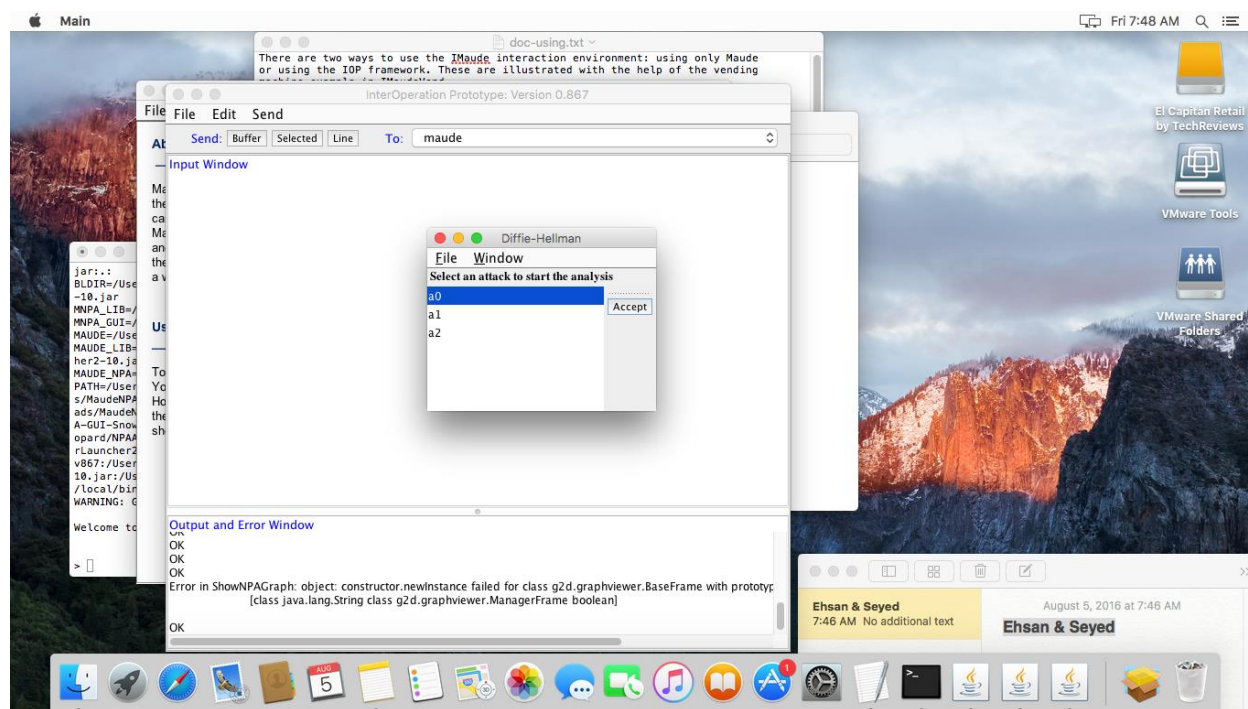
```

شکل ۷: اجرای دستور red initials و red summary و دریافت کد حمله



شکل ۸: نمایی از ابزار Maude-NPA در سیستم عامل OSX

برای کار با ابزار Maude-NPA یک ابزار دارای رابط کاربری معرفی شده است که فقط در سیستم عامل OSX (مک) قابل اجرا است. از آنجایی که ابزار Maude-PSL هم از Maude-NPA استفاده می‌کند برای راحتی در استفاده سیستم عامل OSX در ماشین مجازی (VMWare) نصب و اجرا شد. همچنین ابزار مربوطه نیز بعد از نصب کتابخانه‌ها و ابزارهای مورد نیاز آن نصب شد. در زیر نمایی از آن دیده می‌شود. متأسفانه ابزار دارای مشکلی بود که در هنگام ساختن گراف و سناریوی حمله با خطا روبه‌رو می‌شد و عملاً ابزار بدون استفاده شد. البته این احتمال نیز وجود دارد که چون سیستم عامل روی ماشین مجازی اجرا شده است این خطاها بروز می‌کند. ولی شواهد نشان‌دهنده وجود خطا در کد ابزار است و فقط بر روی نسخه‌های خاصی از این سیستم‌عامل کار می‌کند.



شکل ۹: اجرای پروتکل دفی-هلمن در ابزار Maude-NPA در سیستم عامل OSX و بروز خطا در آن

## ۴ زبان Maude-PSL

زبان Maude-PSL زبانی مبتنی بر ابزار Maude-NPA است. این زبان نشانه‌گذاری<sup>۱۸</sup> آلیس-باب را گسترش داده است. در ادامه نحوه تعریف یک پروتکل به وسیله این زبان بیان می‌شود. سپس اجرای پروتکلی که با این زبان توصیف شده است را نشان می‌دهیم. در پایان توصیف پروتکل Kerberos نسخه ۵ را با این زبان بیان می‌کنیم.

توصیف هر پروتکل در این زبان از ۴ قسمت اصلی تشکیل شده است. Theory، Intruder، Protocol و Attacks. قسمت اول برابری‌ها و معادلات و توابع مختلف مثل ترجمه رمز در آن تعریف می‌شود در قسمت دوم توصیف پروتکل می‌آید. در قسمت سوم توانایی‌های مهاجم توصیف می‌شود و در نهایت در قسمت چهارم حملات مختلف مثل حمله افشای سر تعریف می‌شود.

```
spec name is
Theory
...
Protocol
...
Intruder
...
Attacks
...
ends
```

### ۴.۱ قسمت Theory

نحوه تعریف متغیرها به شکل زیر است. متغیرها فقط در قسمت Theory قابل تعریف هستند و سایر قسمت‌ها از آن استفاده می‌کنند.

```
vars AName BName : Name .
vars N M : Msg .
```

---

<sup>۱۸</sup> Notation

قسمت Theory برای تعریف مفاهیم معماشناسی است. در این قسمت نوع‌هایی که در طول توصیف از آنها استفاده خواهد شد تعریف می‌شوند. برای تعریف نوع از type, types, subtype, subtypes استفاده می‌شود برای مثال:

```
types Name Nonce MultipliedNonces Generator Exp Key GeneratorOrExp Secret .
subtypes Generator Exp < GeneratorOrExp .
subtype Exp < Key .
```

برای تعریف توابع مختلف هم از op, ops استفاده می‌شود برای مثال:

```
ops sec n : Name Fresh -> Secret .
ops a b i : -> Name .
ops e d : Key Msg -> Msg .
```

که برای مثال e برای رمزکردن و d برای ترجمه رمز است. البته در این جا صرفاً نام‌گذاری توابع است و برای تعریف تابع، باید معادله در ادامه همین قسمت نوشته شود. همچنین دو نوع تابع قابل تعریف است توابع به شکل پیشوندی و یا میانوندی که مثال‌های بالا پیشوندی هستند و مثال زیر میانوندی که برای تعریف الحاق<sup>۱۹</sup> استفاده می‌شود.

```
op _;_ : Msg Msg -> Msg .
```

زیرخط‌ها نشان‌دهنده محل قرار گیری عملوندها هستند. برای هر عملگر می‌توان خاصیت جابجایی (comm) یا شرکت‌پذیری (assoc) تعریف کرد. یا از gather(E e) برای شرکت‌پذیری از چپ و gather(e E) برای شرکت‌پذیری از راست استفاده کرد.

```
op _;_ : Msg Msg -> Msg [ gather (e E) ] .
```

برای استفاده از این ویژگی‌ها برای یک عملگر باید هر دو عملوند از یک نوع باشند. برای تعریف عملیات عملگرها و یا عملگرهای جبری مثل توان رسانی از eq استفاده می‌شود.

```
var G : Gen . vars Y Z : MultipliedNonces .
eq exp (exp (G, Y), Z) = exp (G, Y * Z) .
```

برای اینکه محاسبه عبارت پایان پذیرد باید عبارت پیچیده‌تر در سمت چپ قرار گیرد.

## ۴,۲ قسمت Protocol

این قسمت خود از چند زیر قسمت تشکیل شده است.

۱. ورودی‌های پروتکل
۲. توصیف پروتکل
۳. خروجی‌های پروتکل
۴. تعاریف برای کوتاه کردن عبارت‌ها (اختیاری)

در هر پروتکلی تعدادی با نقش‌های مختلف مثل کارخواه یا کارساز وجود دارند برای تعریف نقش‌ها از Roles استفاده می‌شود.

roles role1 role2 ... roleN .

roles A B .

ورودی‌های پروتکل که همان دانسته‌های هر نقش است با In تعریف می‌شود.

vars ANAME BNAME : Name .

In(A) = ANAME , BNAME .

In(B) = BNAME .

در مرحله بعد خود پروتکل توصیف می‌شود برای مثال اگر پروتکل دفی-هلمن که در زیر آمده است را در نظر بگیریم توصیف پروتکل به زبان Maude-PSL در ادامه آمده است:

1.  $A \rightarrow B : A; B; g^{P^*}$
2.  $B \rightarrow A : A; B; g^{P^*}$
3.  $A \rightarrow B : e((g^{P^*})^{P^*}, s)$

شکل ۱۰: پروتکل دفی-هلمن

vars AName BName A1Name : Name .

vars r1 r2 r3 : Fresh .

vars XEA XEB : Exp .

var S : Secret .

1 . A -> B : ANAME ; BNAME ; exp (g, n(ANAME , r1 )) |- A1NAME ; BNAME ; XEB .

2 . B -> A : A1NAME ; BNAME ; exp (g, n(BNAME , r2 )) |- ANAME ; BNAME ; XEA .

3 . A -> B : e(exp(XEA , n(ANAME , r1 )), sec(ANAME , r3 )) |- e(exp(XEB , n(BNAME , r2 )), S) .

تفاوت اصلی این توصیف با توصیف آلیس-باب در این است که هر گذر از پروتکل با دو دیدگاه<sup>۲۰</sup> مطرح می‌شود: دیدگاه ارسال کننده و دریافت کننده. برای مثال در گذر اول ANAME نامی است که B دریافت کرده که می‌تواند نام A باشد یا نباشد. همچنین دیدگاه‌ها توانایی هر طرف را از دیدن مقادیر دریافتی مشخص می‌کند برای مثال در گذر اول A عبارت exp (g, n(ANAME , r1 )) را ارسال می‌کند. از دیدگاه B این عبارت تنها یک مقدار دلخواه است و از محتویات آن بی‌خبر است.

مقادیر خروجی پروتکل که مقادیر مهمی هستند که در طول پروتکل می‌توانند تولید شده باشند را با Out نشان می‌دهند.

Out(A) = exp(XEA , n(A, r1 )) .  
Out(B) = exp(XEB , n(B, r2 )) .

قسمت آخر تعاریف هستند که برای تمیزتر و خوانایی بیشتر پروتکل استفاده می‌شود. برای هر نقش می‌توان آن را تعریف کرد. هر تعریف یک شناسه دارد که ممکن است در تعاریف دیگر هم استفاده شود. همچنین ترتیب هم در آنها تاثیر ندارد.

Def(A) = pa := n(A, r1),  
s := sec(A, r'),  
g^pa := exp(g, pa),  
xea ^pa := exp (XEA , pa) .  
Def(B) = g^pb := exp(g, pb),  
pb := n(B, r2),  
xeb^pb := exp(XEB , pb) .

1 . A -> B : A ; B ; g^pa |- A1 ; B ; XEB .  
2 . B -> A : A1 ; B ; g^pb |- A ; B ; XEA .  
3 . A -> B : e(xea^pa , secret ) |- e(xeb^pb , S) .

## ۴,۳ قسمت Intruder

در این قسمت توانایی‌های مهاجم تعریف می‌شود. فرض می‌شود مهاجم کنترل کامل بر شبکه دارد و می‌تواند پیام‌ها را تغییر دهد، خراب کند یا جابه‌جا کند. برای مثال برای پروتکل دفی-هلمن داریم:

```

var r : Fresh . var P : Name .
vars M1 M2 : Msg . vars NS1 NS2 : Nonce .
var K : Key . var GE : GeneratorOrExp .
K, M1 => e(K, M1), d(K, M1) .
NS1 , NS2 => NS1 * NS2 .
GE , NS1 => exp(GE , NS1) .
M1 ; M2 <=> M1 , M2 .
=> n(i, r), g, P .

```

مثلا برای  $K, M1 \Rightarrow e(K, M1), d(K, M1)$  یعنی اگر مهاجم  $K$  و  $M1$  را بداند می‌تواند  $M1$  را رمز یا ترجمه کند.  $\Rightarrow n(i, r), g, P$  هم یعنی مهاجم توانایی تولید  $p, g$  و  $n(i, r)$  را دارد.

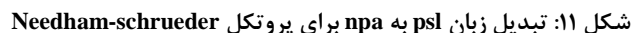
## ۴،۴ قسمت Attacks

در قسمت ترتیبی از حملات شماره‌گذاری شده می‌آید. هر الگوی حمله سناریویی را مطرح می‌کند که باید ثابت شود، پروتکل در برابر آن امن است. برای مثال برای دفی-هلمن داریم:

- 0 .  
 B executes protocol .  
 $\text{Subst}(B) = A1Name \mid\rightarrow a, BName \mid\rightarrow b, S \mid\rightarrow \text{sec}(a, r')$  .  
 without :  
 A executes protocol .  
 $\text{Subst}(A) = AName \mid\rightarrow a, BName \mid\rightarrow b$  .
- 1 .  
 B executes protocol .  
 $\text{Subst}(B) = A1Name \mid\rightarrow a, BName \mid\rightarrow b, S \mid\rightarrow \text{sec}(a, r')$  .  
 Intruder learns  $\text{sec}(a, r')$  .
- 2 .  
 B executes protocol .  
 $\text{Subst}(B) = A1Name \mid\rightarrow a, S \mid\rightarrow \text{sec}(a, r')$  .  
 With constraints  $BName \neq a$  .  
 Intruder learns  $\text{sec}(a, r')$  .

حمله اول حمله بر علیه تصدیق اصالت و حمله دوم بر علیه محرمانگی سر است. متغیرها همان‌هایی هستند که در بخش Theory تعریف شده بودند. «B executes protocol» یعنی باب نیمه پروتکل خود را اجرا می‌کند. « $\text{Subst}(B) = A1Name \mid\rightarrow a, BName \mid\rightarrow b, S \mid\rightarrow \text{sec}(a, r')$ » یعنی متغیرهای مربوط به باب با متغیرهای مشخص شده جابه‌جا می‌شوند. «Intruder learns  $\text{sec}(a, r')$ » آنچه را که مهاجم می‌فهمد را نشان می‌دهد. عبارت without برای اینکه مشخص شود چه چیزی نباید اجرا شود، استفاده می‌شود.





## ۴,۵ درستی سنجی پروتکل Kerberos نسخه ۵ با استفاده از Maude-PSL

```
subtype Masterkey Sessionkey < Key .
```

```
subtype SName UName < Name .

subtype Name Number < Public .

op n : Name Fresh -> Nonce . // Nonce operator

op t : Name Fresh -> Nonce . // Timestamp operator

ops c s i : -> UName [ctor] . // Principals (c is Client) (s is Service) (i is Intruder)

ops kas tgs : -> SName [ctor] . // Principal (kas is KeyAuthenticationServer) (tgs is
TicketGrantingServer)

op mkey : Name Name -> Masterkey . //

op seskey : Name Name Nonce -> Sessionkey .

op e : Key Msg -> Msg . // Encryption operator

op d : Key Msg -> Msg . // Decryption operator

op _;_ : Msg Msg -> Msg [gather (e E)] . // Concatnation

op plus : Fresh Number -> Fresh .

op PlusTime : Nonce -> Nonce . // plusTime T1 = T1+1

// Definition of Encryption and Decryption

var K : Key .

var Z : Msg .

eq d(K, e(K, Z)) = Z .

eq e(K, d(K, Z)) = Z .

var rr : Fresh .

var XX : Name .

var one : Number .

eq t(XX, plus(rr, one)) = PlusTime(t(XX, rr)) .

// Protocol Section
```

## Protocol

```
//Variables

vars CNAME SNAME CviaKAS CviaTGS SviaTGS CviaTGSviaS : UName . //client_Name
Service_Name client-KAS client-TGS service-KAS service-TGS client-service

vars KASNAME TGSNAME TGSviaKAS : SName .

vars r r1 r2 r2' r3 r4 r5 r5' r6 r7 r8 : Fresh .

vars TSC TSS TSKAS TSTGS : Nonce . //client_Timestamp service_Timestamp KAS_Timestamp
TGS_Timestamp

vars nonce' nonce2' : Nonce .

var K : Key .

vars KctgsViaC KctgsViaTGS : Key .

vars ClientKey TGSKey ServiceKey : Masterkey .

//vars kcs' kcs2 : SessionKey .

vars M X Y : Msg .

roles C S KAS TGS . // client service KAS TGS

// Inputs of Principals

In(C) = CNAME, SNAME, KASNAME, TGSNAME . // client_name service_name KAS_name
TGS_name

In(S) = SNAME, TGSNAME . // service_name

In(KAS) = KASNAME . // KAS_name

In(TGS) = TGSNAME, KASNAME . //TGS_name

//Definitions for simplicity

Def(C) = nonce := n(CNAME, r1), nonce2 := n(CNAME, r6), T1 := e(KctgsViaC, t(CNAME, r2)),
T2 := e(kcs', t(CNAME, r5)), ClientKey := mkey(CNAME, KASNAME), kcs' := seskey(CNAME,
SNAME, n(TGSNAME, r7)) .
```

Def(TGS) = TGSKey := mkey(TGSNAME, KASNAME), ServiceKey2 := mkey(SviaTGS, TGSNAME),  
kcs := seskey(CviaTGS, SviaTGS, n(TGSNAME, r7)), tsTGS := t(TGSNAME, r3) .

Def(KAS) = ClientKey2 := mkey(CviaKAS, KASNAME), TGSKey2 := mkey(TGSviaKAS,  
KASNAME), kctgs := seskey(CviaKAS, TGSviaKAS, n(KASNAME, r)), tsKAS := t(KASNAME, r4) .

Def (S) = ServiceKey := mkey(SNAME, TGSNAME), kcs2 := seskey(CviaTGSviaS, SNAME,  
n(TGSNAME, r7)) .

// Protocol Description by Alice-Bob Notation

1 . C -> KAS : CNAME ; TGSNAME ; nonce | - CviaKAS ; TGSviaKAS ; nonce' .

2 . KAS -> C : e(ClientKey2, kctgs ; TGSviaKAS ; nonce') ; e(TGSKey2, CviaKAS ; tsKAS ;  
kctgs) | - e(ClientKey, KctgsViaC ; TGSNAME ; nonce) ; M .

3 . C -> TGS : T1 ; M ; CNAME ; SNAME ; nonce2 | - e(KctgsViaTGS, t(CviaTGS, r2')) ;  
e(TGSKey, CviaTGS ; tsKAS ; kctgs) ; CviaTGS ; SviaTGS ; nonce2' .

4 . TGS -> C : e(KctgsViaTGS, kcs ; SviaTGS ; nonce2') ; e(ServiceKey2, CviaTGS ; tsTGS ;  
kcs) | - e(KctgsViaC, kcs' ; SNAME ; nonce2) ; X .

5 . C -> S : e(kcs', t(CNAME, r5)) ; X | - e(kcs2, t(CviaTGSviaS, r5')) ; e(ServiceKey,  
CviaTGSviaS ; Y ; kcs2) .

6 . S -> C : e(kcs2, PlusTime(d(kcs2, e(kcs2, t(Y, r5'))))) | - e(kcs', PlusTime(d(kcs', T2))) .

//Outputs of Prinipals

Out(C) = kcs', PlusTime(d(kcs', T2)) .

Out(S) = kcs2, PlusTime(d(kcs2, e(kcs2, t(Y, r5'))))) .

Out(KAS) = tsKAS .

Out(TGS) = kctgs, tsTGS .

// Intruder Section

Intruder

var D : Name .

var S : SName .

var r : Fresh .

var K : Key .

var tt : Fresh .

var XX : Name .

vars M N : Msg .

=> D, n(i, r), t(i, r) .

=> mkey(i, D), mkey(D, i) , seskey(i, D, n(i, r)), mkey(i, S) , mkey(S, i) .

=> PlusTime(t(XX, tt)) .

K, M => d(K, M), e(K, M) .

M, N <=> M ; N .

// Attack Section

Attacks

0 .

C executes protocol .

Subst(C) = CNAME |-> c , SNAME |-> s, KASNAME |-> kas , TGSNAME |-> tgs .

1 .

C executes protocol .

Subst(C) = CNAME |-> c , SNAME |-> s, KASNAME |-> kas , TGSNAME |-> tgs .

Intruder learns kcs' .

2 .

KAS executes protocol .

Subst(KAS) = KASNAME |-> kas .

Intruder learns tsKAS .

3 .

TGS executes protocol .

Subst(TGS) = KASNAME |-> kas, TGSNAME |-> tgs .

Intruder learns kctgs .

4 .

TGS executes protocol .

Subst(TGS) = KASNAME |-> kas, TGSNAME |-> tgs .

Intruder learns tsTGS .

5 .

S executes protocol .

Subst(S) = SNAME |-> s, TGSNAME |-> tgs .

Intruder learns kcs2 .

ends

## ۴,۶ درستی‌سنجی پروتکل Kerberos نسخه ۱ با استفاده از Maude-PSL

spec KerberosV1 is

// Theory Section

Theory

types UName SName Name Key Nonce Number Masterkey Sessionkey .

subtype Masterkey Sessionkey < Key .

subtype SName UName < Name .

subtype Name Number < Public .

op n : Name Fresh -> Nonce . // Nonce operator

op t : Name Fresh -> Nonce . // Timestamp operator

ops a b i : -> UName [ctor] . // Principals (a is Alice) (b is Bob) (i is Intruder)

op trusted : -> SName [ctor] . // Principal

op mkey : Name Name -> Masterkey .

op seskey : Name Name Nonce -> Sessionkey .

op e : Key Msg -> Msg . // Encryption operator

op d : Key Msg -> Msg . // Decryption operator

op \_;\_ : Msg Msg -> Msg [gather (e E)] . // Concatnation

op plus : Fresh Number -> Fresh .

op PlusTime : Nonce -> Nonce . // plusTime T1 = T1+1

//op Split : Msg -> Msg . // Split concatenation

// Definition of Encryption and Decryption

var K : Key .

var Z : Msg .

eq d(K, e(K, Z)) = Z .

eq e(K, d(K, Z)) = Z .

//vars G, F : Msg .

//eq F = Split(G ; F) .

var rr : Fresh .

var XX : Name .

var one : Number .

eq t(XX, plus(rr, one)) = PlusTime(t(XX, rr)) .

// Protocol Section

Protocol

//Variables

---

```

vars ANAME BNAME AviaTRUSTED BviaTRUSTED AviaB : UName .

var TRUSTEDNAME : SName . //TGSviaKAS

vars r r1 r2 r2' r3 r3' r4 r5 r5' r6 r7 r8 : Fresh .

var K : Key .

vars ATKEY TrustedKey BTKEY : Masterkey .

//vars ks ks2 ks' : SessionKey .

vars M X Y : Msg .

roles A B TRUSTED . // client service KAS TGS

// Inputs of Principals

In(A) = ANAME, BNAME, TRUSTEDNAME .

In(B) = BNAME, TRUSTEDNAME . // service_name

In(TRUSTED) = TRUSTEDNAME . //TGS_name

//Definitions for simplicity

Def(A) = ATKEY2 := mkey(ANAME, TRUSTEDNAME) .

Def (B) = BTKEY2 := mkey(BNAME, TRUSTEDNAME) .

// Protocol Description by Alice-Bob Notation

1 . A -> TRUSTED : ANAME ; BNAME |- AviaTRUSTED ; BviaTRUSTED .

2 . TRUSTED -> A : e(mkey(AviaTRUSTED, TRUSTEDNAME),
seskey(AviaTRUSTED, BviaTRUSTED, n(TRUSTEDNAME,r7))) ; BviaTRUSTED ;
t(TRUSTEDNAME, r)) ; e(mkey(BviaTRUSTED, TRUSTEDNAME), seskey(AviaTRUSTED,
BviaTRUSTED, n(TRUSTEDNAME, r7))) ; AviaTRUSTED ; t(TRUSTEDNAME, r2)) |-
e(ATKEY2, seskey(ANAME, BNAME, n(TRUSTEDNAME, r8))) ; BNAME ;
t(TRUSTEDNAME, r6)) ; M .

3 . A -> B : M ; e(seskey(ANAME, BNAME, n(TRUSTEDNAME, r8)), ANAME ;
t(ANAME, r3)) |- e(BTKEY2, seskey(AviaB, BNAME, n(TRUSTEDNAME, r5'))) ; AviaB ;
t(TRUSTEDNAME, r5)) ; e(seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')), AviaB ;
t(AviaB, r3')) .

```



---

```
4 . B -> A : e(seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')),
PlusTime(d(seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')), e(seskey(AviaB, BNAME,
n(TRUSTEDNAME, r5')), t(AviaB, r3'))))) |- e(seskey(ANAME, BNAME, n(TRUSTEDNAME,
r8)),PlusTime(t(ANAME, r3)))
```

//Outputs of Prinipals

Out(A) = seskey(ANAME, BNAME, n(TRUSTEDNAME, r8)), PlusTime(t(ANAME, r3)) .

Out(B) = seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')), PlusTime(d(seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')), e(seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')), AviaB ; t(AviaB, r3')))) .

Out(TRUSTED) = seskey(AviaTRUSTED, BviaTRUSTED, n(TRUSTEDNAME, r7)) .

// Intruder Section

Intruder

var D : Name .

var S : SName .

var r : Fresh .

var K : Key .

var tt : Fresh .

var XX : Name .

vars M N : Msg .

=> D, n(i, r), t(i, r) .

=> mkey(i, D), mkey(D, i) , seskey(i, D, n(i, r)), mkey(i, S) , mkey(S, i) .

=> PlusTime(t(XX, tt)) .

K, M => d(K, M), e(K, M) .

M, N <=> M ; N .

// Attack Section

Attacks

0 .

A executes protocol .

Subst(A) = ANAME  $\rightarrow$  a , BNAME  $\rightarrow$  b, TRUSTEDNAME  $\rightarrow$  trusted .

1 .

A executes protocol .

Subst(A) = ANAME  $\rightarrow$  a , BNAME  $\rightarrow$  b, TRUSTEDNAME  $\rightarrow$  trusted .

Intruder learns seskey(ANAME, BNAME, n(TRUSTEDNAME, r8)) .

2 .

TRUSTED executes protocol .

Subst(TRUSTED) = TRUSTEDNAME  $\rightarrow$  trusted .

Intruder learns seskey(AviaTRUSTED, BviaTRUSTED, n(TRUSTEDNAME, r7)) .

3 .

B executes protocol .

Subst(B) = BNAME  $\rightarrow$  b, TRUSTEDNAME  $\rightarrow$  trusted .

Intruder learns seskey(AviaB, BNAME, n(TRUSTEDNAME, r5')) .

ends

## ۵ مراجع

- [۱] Tobias Nipkow, Johannes Hölzl. "Isabelle". Isabelle.in.tum.de. N.p., 2016. Web. 5 Aug. 2016.
- [۲] "PRISM - Probabilistic Symbolic Model Checker". Prismmodelchecker.org. N.p., 2016. Web. 5 Aug. 2016.
- [۳] "Maude-NPA". Maude.cs.uiuc.edu. N.p., 2016. Web. 5 Aug. 2016.
- [۴] "The Real-Time Maude Tool". Heim.ifi.uio.no. N.p., 2016. Web. 5 Aug. 2016.
- [۵] Grimeland, Martin. "Modeling and analysis of time-dependent security protocols in Real-Time Maude." Master's thesis, Dept. of Informatics, University of Oslo (June 2006) (2006).
- [۶] Akbarzadeh, Mojtaba, and Mohammad Abdollahi Azgomi. "A framework for probabilistic model checking of security protocols using coloured stochastic activity networks and pdetool." Telecommunications (IST), 2010 5th International Symposium on. IEEE, 2010.
- [۷] Bellovin, Steven M., and Michael Merritt. "Limitations of the Kerberos authentication system." ACM SIGCOMM Computer Communication Review 20.5 (1990): 119-132.

## ۶ پیوست‌ها

### ۶,۱ کد تحلیل پروتکل Kerberos

#### نسخه ۵ با ابزار Isabelle

```
(* Title:    HOL/Auth/KerberosV.thy

   Author:    Giampaolo Bella, Catania University

*)

section‹The Kerberos Protocol, Version V›

theory KerberosV imports Public begin

text‹The "u" prefix indicates theorems referring to an updated version of
the protocol. The "r" suffix indicates theorems where the confidentiality
assumptions are relaxed by the corresponding arguments.›

abbreviation

  Kas :: agent where

  "Kas == Server"

abbreviation

  Tgs :: agent where

  "Tgs == Friend 0"

axiomatization where

  Tgs_not_bad [iff]: "Tgs ∉ bad"

  —‹Tgs is secure --- we already know that Kas is secure›

definition

  (* authKeys are those contained in an authToken *)

  authKeys :: "event list => key set" where

  "authKeys evs = {authK. ∃A Peer Ta.

    Says Kas A ⟨Crypt (shrK A) ⟨Key authK, Agent Peer, Ta⟩,

      Crypt (shrK Peer) ⟨Agent A, Agent Peer, Key authK, Ta⟩

    ⟩ ∈ set evs}"

definition

  (* A is the true creator of X if she has sent X and X never appeared on
the trace before this event. Recall that traces grow from head. *)

  Issues :: "[agent, agent, msg, event list] => bool"

  ("_ Issues _ with _ on _") where
```

```
"A Issues B with X on evs =

  (∃Y. Says A B Y ∈ set evs ∧ X ∈ parts {Y} ∧

    X ∉ parts (spies (takeWhile (% z. z ≠ Says A B Y) (rev evs))))"
```

```
consts

  (*Duration of the authentication key*)

  authKlife :: nat

  (*Duration of the service key*)

  servKlife :: nat

  (*Duration of an authenticator*)

  authlife :: nat

  (*Upper bound on the time of reaction of a server*)

  replylife :: nat

specification (authKlife)

  authKlife_LB [iff]: "2 ≤ authKlife"

  by blast

specification (servKlife)

  servKlife_LB [iff]: "2 + authKlife ≤ servKlife"

  by blast

specification (authlife)

  authlife_LB [iff]: "Suc 0 ≤ authlife"

  by blast

specification (replylife)

  replylife_LB [iff]: "Suc 0 ≤ replylife"

  by blast

abbreviation

  (*The current time is just the length of the trace!*)

  CT :: "event list=>nat" where

  "CT == length"

abbreviation

  expiredAK :: "[nat, event list] => bool" where

  "expiredAK T evs == authKlife + T < CT evs"

abbreviation

  expiredSK :: "[nat, event list] => bool" where

  "expiredSK T evs == servKlife + T < CT evs"

abbreviation
```

expiredA :: "[nat, event list] => bool" where	]
"expiredA T evs == authlife + T < CT evs"	⇒ Says A Tgs {authTicket,
abbreviation	(Crypt authK {Agent A, Number (CT evs3)}),
valid :: "[nat, nat] => bool" ("valid _ wrt _") where	Agent B} # evs3 ∈ kerbV"
"valid T1 wrt T2 == T1 <= replylife + T2"	(*Unlike version IV, servTicket is not re-encrypted*)
(*-----*)	KV4: "[ evs4 ∈ kerbV; Key servK ∉ used evs4; servK ∈ symKeys;
(* Predicate formalising the association between authKeys and servKeys *)	B ≠ Tgs; authK ∈ symKeys;
definition AKcryptSK :: "[key, key, event list] => bool" where	Says A' Tgs {
"AKcryptSK authK servK evs ==	(Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
∃ A B tt.	Number Ta}),
Says Tgs A {Crypt authK {Key servK, Agent B, tt},	(Crypt authK {Agent A, Number T2}), Agent B}
Crypt (shrK B) {Agent A, Agent B, Key servK, tt} }	∈ set evs4;
∈ set evs"	¬ expiredAK Ta evs4;
inductive_set kerbV :: "event list set"	¬ expiredA T2 evs4;
where	servKlife + (CT evs4) <= authKlife + Ta
Nil: "[] ∈ kerbV"	]
Fake: "[ evsf ∈ kerbV; X ∈ synth (analz (spies evsf)) ]	⇒ Says Tgs A {
⇒ Says Spy B X # evsf ∈ kerbV"	Crypt authK {Key servK, Agent B, Number (CT evs4)},
(*Authentication phase*)	Crypt (shrK B) {Agent A, Agent B, Key servK, Number (CT evs4)}
KV1: "[ evs1 ∈ kerbV ]	} # evs4 ∈ kerbV"
⇒ Says A Kas {Agent A, Agent Tgs, Number (CT evs1)} # evs1	(*Service phase*)
∈ kerbV"	KV5: "[ evs5 ∈ kerbV; authK ∈ symKeys; servK ∈ symKeys;
(*Unlike version IV, authTicket is not re-encrypted*)	A ≠ Kas; A ≠ Tgs;
KV2: "[ evs2 ∈ kerbV; Key authK ∉ used evs2; authK ∈ symKeys;	Says A Tgs
Says A' Kas {Agent A, Agent Tgs, Number T1} ∈ set evs2 ]	{authTicket, Crypt authK {Agent A, Number T2},
⇒ Says Kas A {	Agent B}
Crypt (shrK A) {Key authK, Agent Tgs, Number (CT evs2)},	∈ set evs5;
Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number (CT evs2)}	Says Tgs' A {Crypt authK {Key servK, Agent B, Number Ts},
} # evs2 ∈ kerbV"	servTicket}
(* Authorisation phase *)	∈ set evs5;
KV3: "[ evs3 ∈ kerbV; A ≠ Kas; A ≠ Tgs;	valid Ts wrt T2 ]
Says A Kas {Agent A, Agent Tgs, Number T1} ∈ set evs3;	⇒ Says A B {servTicket,
Says Kas' A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},	Crypt servK {Agent A, Number (CT evs5)} }
authTicket} ∈ set evs3;	# evs5 ∈ kerbV"
valid Ta wrt T1	KV6: "[ evs6 ∈ kerbV; B ≠ Kas; B ≠ Tgs;

```

Says A' B {
  (Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}),
  (Crypt servK {Agent A, Number T3})}

∈ set evs6;

¬ expiredSK Ts evs6;

¬ expiredA T3 evs6
}

⇒ Says B A (Crypt servK (Number Ta2))

# evs6 ∈ kerbV"

(* Leaking an authK... *)

| Oops1: "[ evsO1 ∈ kerbV; A ≠ Spy;

  Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}},

  authTicket} ∈ set evsO1;

  expiredAK Ta evsO1 ]

⇒ Notes Spy {Agent A, Agent Tgs, Number Ta, Key authK}

# evsO1 ∈ kerbV"

(*Leaking a servK... *)

| Oops2: "[ evsO2 ∈ kerbV; A ≠ Spy;

  Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}},

  servTicket} ∈ set evsO2;

  expiredSK Ts evsO2 ]

⇒ Notes Spy {Agent A, Agent B, Number Ts, Key servK}

# evsO2 ∈ kerbV"

declare Says_imp_knows_Spy [THEN parts.Inj, dest]

declare parts.Body [dest]

declare analz_into_parts [dest]

declare Fake_parts_insert_in_Un [dest]

subsection«Lemmas about lists, for reasoning about Issues»

lemma spies_Says_rev: "spies (evs @ [Says A B X]) = insert X (spies evs)"

apply (induct_tac "evs")

apply (rename_tac [2] a b)

apply (induct_tac [2] "a", auto)

done

lemma spies_Gets_rev: "spies (evs @ [Gets A X]) = spies evs"

apply (induct_tac "evs")

```

```

apply (rename_tac [2] a b)

apply (induct_tac [2] "a", auto)

done

lemma spies_Notes_rev: "spies (evs @ [Notes A X]) =

  (if A:bad then insert X (spies evs) else spies evs)"

apply (induct_tac "evs")

apply (rename_tac [2] a b)

apply (induct_tac [2] "a", auto)

done

lemma spies_evs_rev: "spies evs = spies (rev evs)"

apply (induct_tac "evs")

apply (rename_tac [2] a b)

apply (induct_tac [2] "a")

apply (simp_all (no_asm_simp) add: spies_Says_rev spies_Gets_rev
spies_Notes_rev)

done

lemmas parts_spies_evs_revD2 = spies_evs_rev [THEN equalityD2, THEN
parts_mono]

lemma spies_takeWhile: "spies (takeWhile P evs) <= spies evs"

apply (induct_tac "evs")

apply (rename_tac [2] a b)

apply (induct_tac [2] "a", auto)

txt«Resembles <used_subset_append> in theory Event.»

done

lemmas parts_spies_takeWhile_mono = spies_takeWhile [THEN
parts_mono]

subsection«Lemmas about @{term authKeys}»

lemma authKeys_empty: "authKeys [] = {}"

by (simp add: authKeys_def)

lemma authKeys_not_insert:

"(∀A Ta akey Peer.

  ev ≠ Says Kas A {Crypt (shrK A) {akey, Agent Peer, Ta}},

  Crypt (shrK Peer) {Agent A, Agent Peer, akey, Ta} })

⇒ authKeys (ev # evs) = authKeys evs"

by (auto simp add: authKeys_def)

lemma authKeys_insert:

"authKeys

```

(Says Kas A $\llbracket$ Crypt (shrK A) $\llbracket$ Key K, Agent Peer, Number Ta $\rrbracket$ ,	$\in \text{set evs} ;$
Crypt (shrK Peer) $\llbracket$ Agent A, Agent Peer, Key K, Number Ta $\rrbracket \rrbracket \# \text{ evs}$	$\text{evs} \in \text{kerbV} \rrbracket \Rightarrow \text{servK} \notin \text{range shrK} \wedge \text{servK} \in \text{symKeys}"$
= insert K (authKeys evs)"	apply (erule rev_mp)
by (auto simp add: authKeys_def)	apply (erule kerbV.induct, auto)
lemma authKeys_simp:	done
"K $\in$ authKeys	(*Spy never sees another agent's shared key! (unless it's lost at start)*)
(Says Kas A $\llbracket$ Crypt (shrK A) $\llbracket$ Key K', Agent Peer, Number Ta $\rrbracket$ ,	lemma Spy_see_shrK [simp]:
Crypt (shrK Peer) $\llbracket$ Agent A, Agent Peer, Key K', Number Ta $\rrbracket \rrbracket \# \text{ evs}$	"evs $\in$ kerbV $\Rightarrow$ (Key (shrK A) $\in$ parts (spies evs)) = (A $\in$ bad)"
$\Rightarrow$ K = K'   K $\in$ authKeys evs"	apply (erule kerbV.induct)
by (auto simp add: authKeys_def)	apply (frule_tac [7] Says_ticket_parts)
lemma authKeysI:	apply (frule_tac [5] Says_ticket_parts, simp_all)
"Says Kas A $\llbracket$ Crypt (shrK A) $\llbracket$ Key K, Agent Tgs, Number Ta $\rrbracket$ ,	apply (blast+)
Crypt (shrK Tgs) $\llbracket$ Agent A, Agent Tgs, Key K, Number Ta $\rrbracket \rrbracket \in \text{set evs}$	done
$\Rightarrow$ K $\in$ authKeys evs"	lemma Spy_analz_shrK [simp]:
by (auto simp add: authKeys_def)	"evs $\in$ kerbV $\Rightarrow$ (Key (shrK A) $\in$ analz (spies evs)) = (A $\in$ bad)"
lemma authKeys_used: "K $\in$ authKeys evs $\Rightarrow$ Key K $\in$ used evs"	by auto
by (auto simp add: authKeys_def)	lemma Spy_see_shrK_D [dest!]:
subsection«Forwarding Lemmas»	" $\llbracket$ Key (shrK A) $\in$ parts (spies evs); evs $\in$ kerbV $\rrbracket \Rightarrow$ A:bad"
lemma Says_ticket_parts:	by (blast dest: Spy_see_shrK)
"Says S A $\llbracket$ Crypt K $\llbracket$ SesKey, B, TimeStamp $\rrbracket$ , Ticket $\rrbracket$	lemmas Spy_analz_shrK_D = analz_subset_parts [THEN subsetD, THEN
$\in \text{set evs} \Rightarrow$ Ticket $\in$ parts (spies evs)"	Spy_see_shrK_D, dest!]
by blast	text«Nobody can have used non-existent keys!»
lemma Says_ticket_analz:	lemma new_keys_not_used [simp]:
"Says S A $\llbracket$ Crypt K $\llbracket$ SesKey, B, TimeStamp $\rrbracket$ , Ticket $\rrbracket$	" $\llbracket$ Key K $\notin$ used evs; K $\in$ symKeys; evs $\in$ kerbV $\rrbracket$
$\in \text{set evs} \Rightarrow$ Ticket $\in$ analz (spies evs)"	$\Rightarrow$ K $\notin$ keysFor (parts (spies evs))"
by (blast dest: Says_imp_knows_Spy [THEN analz.Inj, THEN analz.Snd])	apply (erule rev_mp)
lemma Oops_range_spies1:	apply (erule kerbV.induct)
" $\llbracket$ Says Kas A $\llbracket$ Crypt KeyA $\llbracket$ Key authK, Peer, Ta $\rrbracket$ , authTicket $\rrbracket$	apply (frule_tac [7] Says_ticket_parts)
$\in \text{set evs} ;$	apply (frule_tac [5] Says_ticket_parts, simp_all)
evs $\in$ kerbV $\rrbracket \Rightarrow$ authK $\notin$ range shrK & authK $\in$ symKeys"	txt«Fake»
apply (erule rev_mp)	apply (force dest!: keysFor_parts_insert)
apply (erule kerbV.induct, auto)	txt«Others»
done	apply (force dest!: analz_shrK_Decrypt)+
lemma Oops_range_spies2:	done
" $\llbracket$ Says Tgs A $\llbracket$ Crypt authK $\llbracket$ Key servK, Agent B, Ts $\rrbracket$ , servTicket $\rrbracket$	(*Earlier, all protocol proofs declared this theorem.

But few of them actually need it! (Another is Yahalom) \*)

lemma new\_keys\_not\_analz:

"[[ $\text{evs} \in \text{kerbV}; K \in \text{symKeys}; \text{Key } K \notin \text{used evs}$ ]]

$\Rightarrow K \notin \text{keysFor } (\text{analz } (\text{spies evs}))$ "

by (blast dest: new\_keys\_not\_used intro: keysFor\_mono [THEN subsetD])

subsection«Regularity Lemmas»

text«These concern the form of items passed in messages»

text«Describes the form of all components sent by Kas»

lemma Says\_Kas\_message\_form:

"[[ Says Kas A {Crypt K {Key authK, Agent Peer, Ta}}, authTicket}

$\in \text{set evs};$

$\text{evs} \in \text{kerbV}$  ]]

$\Rightarrow \text{authK} \notin \text{range shrK} \wedge \text{authK} \in \text{authKeys evs} \wedge \text{authK} \in \text{symKeys} \wedge$

$\text{authTicket} = (\text{Crypt } (\text{shrK Tgs}) \{ \text{Agent A, Agent Tgs, Key authK, Ta} \}) \wedge$

$K = \text{shrK A} \wedge \text{Peer} = \text{Tgs}$ "

apply (erule rev\_mp)

apply (erule kerbV.induct)

apply (simp\_all (no\_asm) add: authKeys\_def authKeys\_insert)

apply blast+

done

(\*This lemma is essential for proving Says\_Tgs\_message\_form:

the session key authK

supplied by Kas in the authentication ticket

cannot be a long-term key!

Generalised to any session keys (both authK and servK).

\*)

lemma SesKey\_is\_session\_key:

"[[ Crypt (shrK Tgs\_B) {Agent A, Agent Tgs\_B, Key SesKey, Number T}

$\in \text{parts } (\text{spies evs}); \text{Tgs\_B} \notin \text{bad};$

$\text{evs} \in \text{kerbV}$  ]]

$\Rightarrow \text{SesKey} \notin \text{range shrK}$ "

apply (erule rev\_mp)

apply (erule kerbV.induct)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, simp\_all, blast)

done

lemma authTicket\_authentic:

"[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}

$\in \text{parts } (\text{spies evs});$

$\text{evs} \in \text{kerbV}$  ]]

$\Rightarrow \text{Says Kas A } \{ \text{Crypt } (\text{shrK A}) \{ \text{Key authK, Agent Tgs, Ta} \},$

$\text{Crypt } (\text{shrK Tgs}) \{ \text{Agent A, Agent Tgs, Key authK, Ta} \} \}$

$\in \text{set evs}$ "

apply (erule rev\_mp)

apply (erule kerbV.induct)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, simp\_all)

txt«Fake, K4»

apply (blast+)

done

lemma authTicket\_crypt\_authK:

"[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}

$\in \text{parts } (\text{spies evs});$

$\text{evs} \in \text{kerbV}$  ]]

$\Rightarrow \text{authK} \in \text{authKeys evs}$ "

by (metis authKeysI authTicket\_authentic)

text«Describes the form of servK, servTicket and authK sent by Tgs»

lemma Says\_Tgs\_message\_form:

"[[ Says Tgs A {Crypt authK {Key servK, Agent B, Ts}}, servTicket}

$\in \text{set evs};$

$\text{evs} \in \text{kerbV}$  ]]

$\Rightarrow B \neq \text{Tgs} \wedge$

$\text{servK} \notin \text{range shrK} \wedge \text{servK} \notin \text{authKeys evs} \wedge \text{servK} \in \text{symKeys} \wedge$

$\text{servTicket} = (\text{Crypt } (\text{shrK B}) \{ \text{Agent A, Agent B, Key servK, Ts} \}) \wedge$

$\text{authK} \notin \text{range shrK} \wedge \text{authK} \in \text{authKeys evs} \wedge \text{authK} \in \text{symKeys}$ "

apply (erule rev\_mp)

apply (erule kerbV.induct)

apply (simp\_all add: authKeys\_insert authKeys\_not\_insert authKeys\_empty  
authKeys\_simp, blast, auto)

txt«Three subcases of Message 4»



apply (blast dest!: authKeys_used Says_Kas_message_form)	Key authK $\notin$ analz (spies evs);
apply (blast dest!: SesKey_is_session_key)	authK $\notin$ range shrK;
apply (blast dest: authTicket_crypt_authK)	evs $\in$ kerbV $\mathbb{I}$
done	$\Rightarrow \exists A \text{ ST. Says Tgs A } \llbracket \text{Crypt authK } \llbracket \text{Key servK, Agent B, Ts} \rrbracket, \text{ST} \rrbracket$
(*	$\in \text{ set evs}$ "
lemma authTicket_form:	apply (erule rev_mp)
lemma servTicket_form:	apply (erule rev_mp)
lemma Says_kas_message_form:	apply (erule kerbV.induct, analz_mono_contra)
lemma Says_tgs_message_form:	apply (frule_tac [7] Says_ticket_parts)
cannot be proved for version V, but a new proof strategy can be used in their	apply (frule_tac [5] Says_ticket_parts, simp_all)
place. The new strategy merely says that both the authTicket and the servTicket	apply blast+
are in parts and in analz as soon as they appear, using lemmas Says_ticket_parts and Says_ticket_analz.	done
The new strategy always lets the simplifier solve cases K3 and K5, saving on	lemma servK_authentic_bis:
long dedicated analyses, which seemed unavoidable. For this reason, lemma	$\mathbb{I} \llbracket \text{Crypt authK } \llbracket \text{Key servK, Agent B, Ts} \rrbracket$
servK_notin_authKeysD is no longer needed.	$\in \text{ parts (spies evs);}$
*)	Key authK $\notin$ analz (spies evs);
subsection«Authenticity theorems: confirm origin of sensitive messages»	B $\neq$ Tgs;
lemma authK_authentic:	evs $\in$ kerbV $\mathbb{I}$
$\mathbb{I} \llbracket \text{Crypt (shrK A) } \llbracket \text{Key authK, Peer, Ta} \rrbracket$	$\Rightarrow \exists A \text{ ST. Says Tgs A } \llbracket \text{Crypt authK } \llbracket \text{Key servK, Agent B, Ts} \rrbracket, \text{ST} \rrbracket$
$\in \text{ parts (spies evs);}$	$\in \text{ set evs}$ "
A $\notin$ bad; evs $\in$ kerbV $\mathbb{I}$	apply (erule rev_mp)
$\Rightarrow \exists A \text{ T. Says Kas A } \llbracket \text{Crypt (shrK A) } \llbracket \text{Key authK, Peer, Ta} \rrbracket, \text{AT} \rrbracket$	apply (erule rev_mp)
$\in \text{ set evs}$ "	apply (erule kerbV.induct, analz_mono_contra)
apply (erule rev_mp)	apply (frule_tac [7] Says_ticket_parts)
apply (erule kerbV.induct)	apply (frule_tac [5] Says_ticket_parts, simp_all, blast+)
apply (frule_tac [7] Says_ticket_parts)	done
apply (frule_tac [5] Says_ticket_parts, simp_all)	text«Authenticity of servK for B»
apply blast+	lemma servTicket_authentic_Tgs:
done	$\mathbb{I} \llbracket \text{Crypt (shrK B) } \llbracket \text{Agent A, Agent B, Key servK, Ts} \rrbracket$
text«If a certain encrypted message appears then it originated with Tgs»	$\in \text{ parts (spies evs); B } \neq \text{ Tgs; B } \notin \text{ bad;}$
lemma servK_authentic:	evs $\in$ kerbV $\mathbb{I}$
$\mathbb{I} \llbracket \text{Crypt authK } \llbracket \text{Key servK, Agent B, Ts} \rrbracket$	$\Rightarrow \exists \text{authK.}$
$\in \text{ parts (spies evs);}$	Says Tgs A $\llbracket \text{Crypt authK } \llbracket \text{Key servK, Agent B, Ts} \rrbracket,$
	Crypt (shrK B) $\llbracket \text{Agent A, Agent B, Key servK, Ts} \rrbracket \rrbracket$
	$\in \text{ set evs}$ "

apply (erule rev_mp)	$\in \text{parts (spies evs); } B \neq \text{Tgs; } B \notin \text{bad;}$
apply (erule kerbV.induct)	$\text{evs} \in \text{kerbV} \text{ ]}$
apply (frule_tac [7] Says_ticket_parts)	$\Rightarrow \exists \text{authK Ta.}$
apply (frule_tac [5] Says_ticket_parts, simp_all, blast+)	Says Kas A
done	$\{\{\text{Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta\}},$
text«Anticipated here from next subsection»	$\text{Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\} \}$
lemma K4_imp_K2:	$\in \text{set evs"}$
"[[ Says Tgs A {\Crypt authK {\Key servK, Agent B, Number Ts}}, servTicket}	by (metis K4_imp_K2 servTicket_authentic_Tgs)
$\in \text{set evs; evs} \in \text{kerbV} \text{ ]}$	lemma u_servTicket_authentic_Kas:
$\Rightarrow \exists \text{Ta. Says Kas A}$	"[[ Crypt (shrK B) {\Agent A, Agent B, Key servK, Number Ts}
$\{\{\text{Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta\}},$	$\in \text{parts (spies evs); } B \neq \text{Tgs; } B \notin \text{bad;}$
$\text{Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\} \}$	$\text{evs} \in \text{kerbV} \text{ ]}$
$\in \text{set evs"}$	$\Rightarrow \exists \text{authK Ta.}$
apply (erule rev_mp)	Says Kas A
apply (erule kerbV.induct)	$\{\{\text{Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta\}},$
apply (frule_tac [7] Says_ticket_parts)	$\text{Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\} \}$
apply (frule_tac [5] Says_ticket_parts, simp_all, auto)	$\in \text{set evs} \wedge$
apply (metis MPair_analz Says_imp_analz_Spy analz_conj_parts authTicket_authentic)	$\text{servKlife} + \text{Ts} \leq \text{authKlife} + \text{Ta}"$
done	by (metis servTicket_authentic_Tgs u_K4_imp_K2)
text«Anticipated here from next subsection»	lemma servTicket_authentic:
lemma u_K4_imp_K2:	"[[ Crypt (shrK B) {\Agent A, Agent B, Key servK, Number Ts}
"[[ Says Tgs A {\Crypt authK {\Key servK, Agent B, Number Ts}}, servTicket} $\in$ set evs; evs $\in$ kerbV ]]	$\in \text{parts (spies evs); } B \neq \text{Tgs; } B \notin \text{bad;}$
$\Rightarrow \exists \text{Ta. Says Kas A} \{\{\text{Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta\}},$	$\text{evs} \in \text{kerbV} \text{ ]}$
$\text{Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\} \}$	$\Rightarrow \exists \text{Ta authK.}$
$\in \text{set evs}$	Says Kas A {\Crypt (shrK A) {\Key authK, Agent Tgs, Number Ta}},
$\wedge \text{servKlife} + \text{Ts} \leq \text{authKlife} + \text{Ta}"$	$\text{Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\} \} \in$ set evs
apply (erule rev_mp)	$\wedge \text{Says Tgs A} \{\{\text{Crypt authK \{Key servK, Agent B, Number Ts\}},$
apply (erule kerbV.induct)	$\text{Crypt (shrK B) \{Agent A, Agent B, Key servK, Number Ts\}} \}$
apply (frule_tac [7] Says_ticket_parts)	$\in \text{set evs"}$
apply (frule_tac [5] Says_ticket_parts, simp_all, auto)	by (metis K4_imp_K2 servTicket_authentic_Tgs)
apply (blast dest!: Says_imp_spies [THEN parts.Inj, THEN parts.Fst, THEN authTicket_authentic])	lemma u_servTicket_authentic:
done	"[[ Crypt (shrK B) {\Agent A, Agent B, Key servK, Number Ts}
lemma servTicket_authentic_Kas:	$\in \text{parts (spies evs); } B \neq \text{Tgs; } B \notin \text{bad;}$
"[[ Crypt (shrK B) {\Agent A, Agent B, Key servK, Number Ts}	$\text{evs} \in \text{kerbV} \text{ ]}$

$\Rightarrow \exists Ta \text{ authK.}$	$\Rightarrow K=K' \wedge B=B' \wedge T=T''$
Says Kas A $\{\{\text{Crypt (shrK A) } \{\{\text{Key authK, Agent Tgs, Number Ta}\}\}$ ,	apply (erule rev_mp)
Crypt (shrK Tgs) $\{\{\text{Agent A, Agent Tgs, Key authK, Number Ta}\}\} \in$	apply (erule rev_mp)
set evs	apply (erule rev_mp)
$\wedge$ Says Tgs A $\{\{\text{Crypt authK } \{\{\text{Key servK, Agent B, Number Ts}\}\}$ ,	apply (erule kerbV.induct, analz_mono_contra)
Crypt (shrK B) $\{\{\text{Agent A, Agent B, Key servK, Number Ts}\}\}$	apply (frule_tac [7] Says_ticket_parts)
$\in$ set evs	apply (frule_tac [5] Says_ticket_parts, simp_all)
$\wedge \text{servKlife} + Ts \leq \text{authKlife} + Ta$ "	txt«Fake, K2, K4»
by (metis servTicket_authentic_Tgs u_K4_imp_K2)	apply (blast+)
lemma u_NotexpiredSK_NotexpiredAK:	done
" $\llbracket \neg \text{expiredSK Ts evs; servKlife} + Ts \leq \text{authKlife} + Ta \rrbracket$	text«This inevitably has an existential form in version V»
$\Rightarrow \neg \text{expiredAK Ta evs}$ "	lemma Says_K5:
by (metis order_le_less_trans)	" $\llbracket \text{Crypt servK } \{\{\text{Agent A, Number T3}\} \in \text{parts (spies evs);}$
subsection«Reliability: friendly agents send something if something else	Says Tgs A $\{\{\text{Crypt authK } \{\{\text{Key servK, Agent B, Number Ts}\}\}$ ,
happened»	
lemma K3_imp_K2:	$\text{servTicket}\} \in \text{set evs;}$
" $\llbracket$ Says A Tgs	$\text{Key servK} \notin \text{analz (spies evs);}$
$\{\{\text{authTicket, Crypt authK } \{\{\text{Agent A, Number T2}\}, \text{Agent B}\}$	$A \notin \text{bad; } B \notin \text{bad; evs} \in \text{kerbV} \rrbracket$
$\in$ set evs;	$\Rightarrow \exists ST. \text{Says A B } \{\{\text{ST, Crypt servK } \{\{\text{Agent A, Number T3}\}\} \in \text{set evs}$
$A \notin \text{bad; evs} \in \text{kerbV} \rrbracket$	apply (erule rev_mp)
$\Rightarrow \exists Ta AT. \text{Says Kas A } \{\{\text{Crypt (shrK A) } \{\{\text{Key authK, Agent Tgs, Ta}\}\}$	apply (erule rev_mp)
$AT\} \in \text{set evs}$ "	apply (erule rev_mp)
apply (erule rev_mp)	apply (erule kerbV.induct, analz_mono_contra)
apply (erule kerbV.induct)	apply (frule_tac [5] Says_ticket_parts)
apply (frule_tac [7] Says_ticket_parts)	apply (frule_tac [7] Says_ticket_parts)
apply (frule_tac [5] Says_ticket_parts, simp_all, blast, blast)	apply (simp_all (no_asm_simp) add: all_conj_distrib)
apply (blast dest: Says_imp_spies [THEN parts.Inj, THEN parts.Fst, THEN	apply blast
authK_authentic])	txt«K3»
done	apply (blast dest: authK_authentic Says_Kas_message_form
text«Anticipated here from next subsection. An authK is encrypted by one	Says_Tgs_message_form)
and only one Shared key. A servK is encrypted by one and only one authK.»	txt«K4»
lemma Key_unique_SesKey:	apply (force dest!: Crypt_imp_keysFor)
" $\llbracket \text{Crypt K } \{\{\text{Key SesKey, Agent B, T}\}$	txt«K5»
$\in \text{parts (spies evs);}$	apply (blast dest: Key_unique_SesKey)
$\text{Crypt K' } \{\{\text{Key SesKey, Agent B', T'}\}$	done
$\in \text{parts (spies evs); Key SesKey} \notin \text{analz (spies evs);}$	text«Anticipated here from next subsection»
$\text{evs} \in \text{kerbV} \rrbracket$	

lemma unique\_CryptKey:

"[[ Crypt (shrK B) (Agent A, Agent B, Key SesKey, T)

∈ parts (spies evs);

Crypt (shrK B') (Agent A', Agent B', Key SesKey, T')

∈ parts (spies evs); Key SesKey ∉ analz (spies evs);

evs ∈ kerbV ]

⇒ A=A' & B=B' & T=T'"

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, simp\_all)

txt⟨Fake, K2, K4⟩

apply (blast+)

done

lemma Says\_K6:

"[[ Crypt servK (Number T3) ∈ parts (spies evs);

Says Tgs A (Crypt authK (Key servK, Agent B, Number Ts),

servTicket) ∈ set evs;

Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; evs ∈ kerbV ]

⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"

apply (frule Says\_Tgs\_message\_form, assumption, clarify)

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts)

apply simp\_all

txt⟨fake⟩

apply blast

txt⟨K4⟩

apply (force dest!: Crypt\_imp\_keysFor)

txt⟨K6⟩

apply (metis MPair\_parts Says\_imp\_parts\_knows\_Spy unique\_CryptKey)

done

text⟨Needs a unicity theorem, hence moved here⟩

lemma servK\_authentic\_ter:

"[[ Says Kas A

(Crypt (shrK A) (Key authK, Agent Tgs, Ta)), authTicket) ∈ set evs;

Crypt authK (Key servK, Agent B, Ts)

∈ parts (spies evs);

Key authK ∉ analz (spies evs);

evs ∈ kerbV ]

⇒ Says Tgs A (Crypt authK (Key servK, Agent B, Ts),

Crypt (shrK B) (Agent A, Agent B, Key servK, Ts))

∈ set evs"

apply (frule Says\_Kas\_message\_form, assumption)

apply clarify

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, simp\_all, blast)

txt⟨K2 and K4 remain⟩

apply (blast dest!: servK\_authentic Says\_Tgs\_message\_form  
authKeys\_used)

apply (blast dest!: unique\_CryptKey)

done

subsection⟨Unicity Theorems⟩

text⟨The session key, if secure, uniquely identifies the Ticket

whether authTicket or servTicket. As a matter of fact, one can read

also Tgs in the place of B.⟩

lemma unique\_authKeys:

"[[ Says Kas A

(Crypt Ka (Key authK, Agent Tgs, Ta), X) ∈ set evs;

Says Kas A'

```

    {Crypt Ka' {Key authK, Agent Tgs, Ta'}, X'} ∈ set evs;

    evs ∈ kerbV ]] ⇒ A=A' ∧ Ka=Ka' ∧ Ta=Ta' ∧ X=X'"

apply (erule rev_mp)

apply (erule rev_mp)

apply (erule kerbV.induct)

apply (frule_tac [7] Says_ticket_parts)

apply (frule_tac [5] Says_ticket_parts, simp_all)

apply blast+

done

text‹servK uniquely identifies the message from Tgs›

lemma unique_servKeys:

  "[[ Says Tgs A

    {Crypt K {Key servK, Agent B, Ts'}, X} ∈ set evs;

    Says Tgs A'

    {Crypt K' {Key servK, Agent B', Ts'}, X'} ∈ set evs;

    evs ∈ kerbV ]] ⇒ A=A' ∧ B=B' ∧ K=K' ∧ Ts=Ts' ∧ X=X'"

apply (erule rev_mp)

apply (erule rev_mp)

apply (erule kerbV.induct)

apply (frule_tac [7] Says_ticket_parts)

apply (frule_tac [5] Says_ticket_parts, simp_all)

apply blast+

done

subsection‹Lemmas About the Predicate @{term AKcryptSK}›

lemma not_AKcryptSK_Nil [iff]: "¬ AKcryptSK authK servK []"

apply (simp add: AKcryptSK_def)

done

lemma AKcryptSK1:

  "[[ Says Tgs A {Crypt authK {Key servK, Agent B, tt}, X} ∈ set evs;

    evs ∈ kerbV ]] ⇒ AKcryptSK authK servK evs"

by (metis AKcryptSK_def Says_Tgs_message_form)

lemma AKcryptSK_Says [simp]:

  "AKcryptSK authK servK (Says S A X # evs) =

    (S = Tgs ∧

    (∃B tt. X = {Crypt authK {Key servK, Agent B, tt},

```

```

    Crypt (shrK B) {Agent A, Agent B, Key servK, tt} ))

    | AKcryptSK authK servK evs)"

by (auto simp add: AKcryptSK_def)

lemma AKcryptSK_Notes [simp]:

  "AKcryptSK authK servK (Notes A X # evs) =

    AKcryptSK authK servK evs"

by (auto simp add: AKcryptSK_def)

(*A fresh authK cannot be associated with any other

(with respect to a given trace). *)

lemma Auth_fresh_not_AKcryptSK:

  "[[ Key authK ∉ used evs; evs ∈ kerbV ]]

  ⇒ ¬ AKcryptSK authK servK evs"

apply (unfold AKcryptSK_def)

apply (erule rev_mp)

apply (erule kerbV.induct)

apply (frule_tac [7] Says_ticket_parts)

apply (frule_tac [5] Says_ticket_parts, simp_all, blast)

done

(*A fresh servK cannot be associated with any other

(with respect to a given trace). *)

lemma Serv_fresh_not_AKcryptSK:

  "Key servK ∉ used evs ⇒ ¬ AKcryptSK authK servK evs"

by (auto simp add: AKcryptSK_def)

lemma authK_not_AKcryptSK:

  "[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, tk}

    ∈ parts (spies evs); evs ∈ kerbV ]]

  ⇒ ¬ AKcryptSK K authK evs"

apply (erule rev_mp)

apply (erule kerbV.induct)

apply (frule_tac [7] Says_ticket_parts)

apply (frule_tac [5] Says_ticket_parts, simp_all)

txt‹Fake,K2,K4›

apply (auto simp add: AKcryptSK_def)

done

text‹A secure serverkey cannot have been used to encrypt others›

```

lemma servK\_not\_AKcryptSK:

"[[ Crypt (shrK B) (Agent A, Agent B, Key SK, tt) ∈ parts (spies evs);

Key SK ∉ analz (spies evs); SK ∈ symKeys;

B ≠ Tgs; evs ∈ kerbV ]]

⇒ ¬ AKcryptSK SK K evs"

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, simp\_all, blast)

txt«K4»

apply (metis Auth\_fresh\_not\_AKcryptSK MPair\_parts  
Says\_imp\_parts\_knows\_Spy authKeys\_used authTicket\_crypt\_authK  
unique\_CryptKey)

done

text«Long term keys are not issued as servKeys»

lemma shrK\_not\_AKcryptSK:

"evs ∈ kerbV ⇒ ¬ AKcryptSK K (shrK A) evs"

apply (unfold AKcryptSK\_def)

apply (erule kerbV.induct)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts, auto)

done

text«The Tgs message associates servK with authK and therefore not with  
any

other key authK.»

lemma Says\_Tgs\_AKcryptSK:

"[[ Says Tgs A (Crypt authK (Key servK, Agent B, tt)), X ]]

∈ set evs;

authK' ≠ authK; evs ∈ kerbV ]]

⇒ ¬ AKcryptSK authK' servK evs"

by (metis AKcryptSK\_def unique\_servKeys)

lemma AKcryptSK\_not\_AKcryptSK:

"[[ AKcryptSK authK servK evs; evs ∈ kerbV ]]

⇒ ¬ AKcryptSK servK K evs"

apply (erule rev\_mp)

apply (erule kerbV.induct)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (frule\_tac [5] Says\_ticket\_parts)

apply (simp\_all, safe)

txt«K4 splits into subcases»

prefer 4 apply (blast dest!: authK\_not\_AKcryptSK)

txt«servK is fresh and so could not have been used, by

«new\_keys\_not\_used»

prefer 2

apply (force dest!: Crypt\_imp\_invKey\_keysFor simp add: AKcryptSK\_def)

txt«Others by freshness»

apply (blast+)

done

lemma not\_different\_AKcryptSK:

"[[ AKcryptSK authK servK evs;

authK' ≠ authK; evs ∈ kerbV ]]

⇒ ¬ AKcryptSK authK' servK evs ∧ servK ∈ symKeys"

apply (simp add: AKcryptSK\_def)

apply (blast dest: unique\_servKeys Says\_Tgs\_message\_form)

done

text«The only session keys that can be found with the help of session keys  
are

those sent by Tgs in step K4.»

text«We take some pains to express the property

as a logical equivalence so that the simplifier can apply it.»

lemma Key\_analz\_image\_Key\_lemma:

"P → (Key K ∈ analz (Key`KK Un H)) → (K:KK | Key K ∈ analz H)

⇒

P → (Key K ∈ analz (Key`KK Un H)) = (K:KK | Key K ∈ analz H)"

by (blast intro: analz\_mono [THEN subsetD])

lemma AKcryptSK\_analz\_insert:

"[[ AKcryptSK K K' evs; K ∈ symKeys; evs ∈ kerbV ]]

⇒ Key K' ∈ analz (insert (Key K) (spies evs))"

apply (simp add: AKcryptSK\_def, clarify)

apply (drule Says\_imp\_spies [THEN analz.Inj, THEN analz\_insertI], auto)

done

lemma authKeys\_are\_not\_AKcryptSK:

"[[ K ∈ authKeys evs Un range shrK; evs ∈ kerbV ]]

⇒ ∀SK. ¬ AKcryptSK SK K evs ∧ K ∈ symKeys"

apply (simp add: authKeys\_def AKcryptSK\_def)

apply (blast dest: Says\_Kas\_message\_form Says\_Tgs\_message\_form)

done

lemma not\_authKeys\_not\_AKcryptSK:

"[[ K ∉ authKeys evs;

K ∉ range shrK; evs ∈ kerbV ]]

⇒ ∀SK. ¬ AKcryptSK K SK evs"

apply (simp add: AKcryptSK\_def)

apply (blast dest: Says\_Tgs\_message\_form)

done

subsection‹Secrecy Theorems›

text‹For the Oops2 case of the next theorem›

lemma Oops2\_not\_AKcryptSK:

"[[ evs ∈ kerbV;

Says Tgs A {Crypt authK

{Key servK, Agent B, Number Ts}, servTicket}

∈ set evs ]]

⇒ ¬ AKcryptSK servK SK evs"

by (blast dest: AKcryptSKI AKcryptSK\_not\_AKcryptSK)

text‹Big simplification law for keys SK that are not crypted by keys in KK

It helps prove three, otherwise hard, facts about keys. These facts are

exploited as simplification laws for analz, and also "limit the damage"

in case of loss of a key to the spy. See ESORICS98.›

lemma Key\_analz\_image\_Key [rule\_format (no\_asm)]:

"evs ∈ kerbV ⇒

(∀SK KK. SK ∈ symKeys & KK <= -(range shrK) →

(∀K ∈ KK. ¬ AKcryptSK K SK evs) →

(Key SK ∈ analz (Key`KK Un (spies evs))) =

(SK ∈ KK | Key SK ∈ analz (spies evs)))"

apply (erule kerbV.induct)

apply (frule\_tac [10] Oops\_range\_spies2)

apply (frule\_tac [9] Oops\_range\_spies1)

(\*Used to apply Says\_tgs\_message form, which is no longer available.

Instead...\*)

apply (drule\_tac [7] Says\_ticket\_analz)

(\*Used to apply Says\_kas\_message form, which is no longer available.

Instead...\*)

apply (drule\_tac [5] Says\_ticket\_analz)

apply (safe del: impl intro!: Key\_analz\_image\_Key\_lemma [THEN impl])

txt‹Case-splits for Oops1 and message 5: the negated case simplifies using  
the induction hypothesis›

apply (case\_tac [9] "AKcryptSK authK SK evsO1")

apply (case\_tac [7] "AKcryptSK servK SK evs5")

apply (simp\_all del: image\_insert

add: analz\_image\_freshK\_simps AKcryptSK\_Says shrK\_not\_AKcryptSK

Oops2\_not\_AKcryptSK Auth\_fresh\_not\_AKcryptSK

Serv\_fresh\_not\_AKcryptSK Says\_Tgs\_AKcryptSK Spy\_analz\_shrK)

txt‹Fake›

apply spy\_analz

txt‹K2›

apply blast

txt‹Cases K3 and K5 solved by the simplifier thanks to the ticket being in

analz - this strategy is new wrt version IV›

txt‹K4›

apply (blast dest!: authK\_not\_AKcryptSK)

txt‹Oops1›

apply (metis AKcryptSK\_analz\_insert insert\_Key\_singleton)

done

text‹First simplification law for analz: no session keys encrypt

authentication keys or shared keys.›

lemma analz\_insert\_freshK1:

"[[ evs ∈ kerbV; K ∈ authKeys evs Un range shrK;

SesKey ∉ range shrK ]]

⇒ (Key K ∈ analz (insert (Key SesKey) (spies evs))) =

(K = SesKey | Key K ∈ analz (spies evs))"

apply (frule authKeys\_are\_not\_AKcryptSK, assumption)

apply (simp del: image\_insert

```

      add: analz_image_freshK_simps add: Key_analz_image_Key)

done

text«Second simplification law for analz: no service keys encrypt any other
keys.»

lemma analz_insert_freshK2:

  "[[ evs ∈ kerbV; servK ∉ {authKeys evs}; servK ∉ range shrK;

    K ∈ symKeys ]]

  ⇒ (Key K ∈ analz (insert (Key servK) (spies evs))) =

    (K = servK | Key K ∈ analz (spies evs))"

apply (frule not_authKeys_not_AKcryptSK, assumption, assumption)

apply (simp del: image_insert

      add: analz_image_freshK_simps add: Key_analz_image_Key)

done

text«Third simplification law for analz: only one authentication key encrypts
a certain service key.»

lemma analz_insert_freshK3:

  "[[ AKcryptSK authK servK evs;

    authK' ≠ authK; authK' ∉ range shrK; evs ∈ kerbV ]]

  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =

    (servK = authK' | Key servK ∈ analz (spies evs))"

apply (drule_tac authK' = authK' in not_different_AKcryptSK, blast,
assumption)

apply (simp del: image_insert

      add: analz_image_freshK_simps add: Key_analz_image_Key)

done

lemma analz_insert_freshK3_bis:

  "[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}

    ∈ set evs;

    authK ≠ authK'; authK' ∉ range shrK; evs ∈ kerbV ]]

  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =

    (servK = authK' | Key servK ∈ analz (spies evs))"

apply (frule AKcryptSK1, assumption)

apply (simp add: analz_insert_freshK3)

done

text«a weakness of the protocol»

lemma authK_compromises_servK:

  "[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}

```

```

    ∈ set evs; authK ∈ symKeys;

    Key authK ∈ analz (spies evs); evs ∈ kerbV ]]

  ⇒ Key servK ∈ analz (spies evs)"

by (metis Says_imp_analz_Spy analz.Fst analz_Decrypt')

text«lemma ‹servK_notin_authKeysD› not needed in version V»

text«If Spy sees the Authentication Key sent in msg K2, then

  the Key has expired.»

lemma Confidentiality_Kas_lemma [rule_format]:

  "[[ authK ∈ symKeys; A ∉ bad; evs ∈ kerbV ]]

  ⇒ Says Kas A

    {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}},

    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}

  ∈ set evs →

  Key authK ∈ analz (spies evs) →

  expiredAK Ta evs"

apply (erule kerbV.induct)

apply (frule_tac [10] Oops_range_spies2)

apply (frule_tac [9] Oops_range_spies1)

apply (frule_tac [7] Says_ticket_analz)

apply (frule_tac [5] Says_ticket_analz)

apply (safe del: impl conjI impCE)

apply (simp_all (no_asm_simp) add: Says_Kas_message_form less_Suc1
analz_insert_eq not_parts_not_analz analz_insert_freshK1 pushes)

txt«Fake»

apply spy_analz

txt«K2»

apply blast

txt«K4»

apply blast

txt«Oops1»

apply (blast dest!: unique_authKeys intro: less_Suc1)

txt«Oops2»

apply (blast dest: Says_Tgs_message_form Says_Kas_message_form)

done

lemma Confidentiality_Kas:

```



```

"[[ Says Kas A
  [[Crypt Ka [[Key authK, Agent Tgs, Number Ta]], authTicket]]
  ∈ set evs;
  ~ expiredAK Ta evs;
  A ∉ bad; evs ∈ kerbV ]]
⇒ Key authK ∉ analz (spies evs)"

apply (blast dest: Says_Kas_message_form Confidentiality_Kas_lemma)

done

text<(If Spy sees the Service Key sent in msg K4, then
  the Key has expired.)

lemma Confidentiality_lemma [rule_format]:

"[[ Says Tgs A
  [[Crypt authK [[Key servK, Agent B, Number Ts]],
    Crypt (shrK B) [[Agent A, Agent B, Key servK, Number Ts]]]]
  ∈ set evs;
  Key authK ∉ analz (spies evs);
  servK ∈ symKeys;
  A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
⇒ Key servK ∈ analz (spies evs) →
  expiredSK Ts evs"

apply (erule rev_mp)

apply (erule rev_mp)

apply (erule kerbV.induct)

apply (rule_tac [9] impl)+

—<The Oops1 case is unusual: must simplify

@{term "Authkey ∉ analz (spies (ev#evs))"}, not letting

<analz_mono_contra> weaken it to

@{term "Authkey ∉ analz (spies evs)"},

for we then conclude @{term "authK ≠ authKa"}.>

apply analz_mono_contra

apply (frule_tac [10] Oops_range_spies2)

apply (frule_tac [9] Oops_range_spies1)

apply (frule_tac [7] Says_ticket_analz)

apply (frule_tac [5] Says_ticket_analz)

apply (safe del: impl conjI impCE)

```

```

apply (simp_all add: less_Suc1 new_keys_not_analz
  Says_Kas_message_form Says_Tgs_message_form analz_insert_eq
  not_parts_not_analz analz_insert_freshK1 analz_insert_freshK2
  analz_insert_freshK3_bis pushes)

txt<Fake>

apply spy_analz

txt<K2>

apply (blast intro: parts_insertI less_Suc1)

txt<K4>

apply (blast dest: authTicket_authentic Confidentiality_Kas)

txt<Oops1>

apply (blast dest: Says_Kas_message_form Says_Tgs_message_form intro:
  less_Suc1)

txt<Oops2>

apply (metis Suc_le_eq linorder_linear linorder_not_le msg.simps(2)
  unique_servKeys)

done

text<(In the real world Tgs can't check wheter authK is secure!)

lemma Confidentiality_Tgs:

"[[ Says Tgs A
  [[Crypt authK [[Key servK, Agent B, Number Ts]], servTicket]]
  ∈ set evs;
  Key authK ∉ analz (spies evs);
  ~ expiredSK Ts evs;
  A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
⇒ Key servK ∉ analz (spies evs)"

by (blast dest: Says_Tgs_message_form Confidentiality_lemma)

text<(In the real world Tgs CAN check what Kas sends!)

lemma Confidentiality_Tgs_bis:

"[[ Says Kas A
  [[Crypt Ka [[Key authK, Agent Tgs, Number Ta]], authTicket]]
  ∈ set evs;
  Says Tgs A
  [[Crypt authK [[Key servK, Agent B, Number Ts]], servTicket]]
  ∈ set evs;
  ~ expiredAK Ta evs; ~ expiredSK Ts evs;
  A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
⇒ Key servK ∉ analz (spies evs)"

```

by (blast dest!: Confidentiality\_Kas Confidentiality\_Tgs)

text«Most general form»

lemmas Confidentiality\_Tgs\_ter = authTicket\_authentic [THEN Confidentiality\_Tgs\_bis]

lemmas Confidentiality\_Auth\_A = authK\_authentic [THEN exE, THEN Confidentiality\_Kas]

text«Needs a confidentiality guarantee, hence moved here.

Authenticity of servK for A»

lemma servK\_authentic\_bis\_r:

"[[ Crypt (shrK A) ⟨Key authK, Agent Tgs, Number Ta⟩

∈ parts (spies evs);

Crypt authK ⟨Key servK, Agent B, Number Ts⟩

∈ parts (spies evs);

¬ expiredAK Ta evs; A ∉ bad; evs ∈ kerbV ]]

⇒ Says Tgs A ⟨Crypt authK ⟨Key servK, Agent B, Number Ts⟩,

Crypt (shrK B) ⟨Agent A, Agent B, Key servK, Number Ts⟩ ]]

∈ set evs"

by (metis Confidentiality\_Kas authK\_authentic servK\_authentic\_ter)

lemma Confidentiality\_Serv\_A:

"[[ Crypt (shrK A) ⟨Key authK, Agent Tgs, Number Ta⟩

∈ parts (spies evs);

Crypt authK ⟨Key servK, Agent B, Number Ts⟩

∈ parts (spies evs);

¬ expiredAK Ta evs; ¬ expiredSK Ts evs;

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ Key servK ∉ analz (spies evs)"

apply (drule authK\_authentic, assumption, assumption)

apply (blast dest: Confidentiality\_Kas Says\_Kas\_message\_form servK\_authentic\_ter Confidentiality\_Tgs\_bis)

done

lemma Confidentiality\_B:

"[[ Crypt (shrK B) ⟨Agent A, Agent B, Key servK, Number Ts⟩

∈ parts (spies evs);

Crypt authK ⟨Key servK, Agent B, Number Ts⟩

∈ parts (spies evs);

Crypt (shrK A) ⟨Key authK, Agent Tgs, Number Ta⟩

∈ parts (spies evs);

¬ expiredSK Ts evs; ¬ expiredAK Ta evs;

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ Key servK ∉ analz (spies evs)"

apply (frule authK\_authentic)

apply (erule\_tac [3] exE)

apply (frule\_tac [3] Confidentiality\_Kas)

apply (frule\_tac [6] servTicket\_authentic, auto)

apply (blast dest!: Confidentiality\_Tgs\_bis dest: Says\_Kas\_message\_form servK\_authentic unique\_servKeys unique\_authKeys)

done

lemma u\_Confidentiality\_B:

"[[ Crypt (shrK B) ⟨Agent A, Agent B, Key servK, Number Ts⟩

∈ parts (spies evs);

¬ expiredSK Ts evs;

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ Key servK ∉ analz (spies evs)"

by (blast dest: u\_servTicket\_authentic u\_NotexpiredSK\_NotexpiredAK Confidentiality\_Tgs\_bis)

subsection«Parties authentication: each party verifies "the identity of

another party who generated some data" (quoted from Neuman and Ts'o).»

text«These guarantees don't assess whether two parties agree on

the same session key: sending a message containing a key

doesn't a priori state knowledge of the key.»

text«These didn't have existential form in version IV»

lemma B\_authenticates\_A:

"[[ Crypt servK ⟨Agent A, Number T3⟩ ∈ parts (spies evs);

Crypt (shrK B) ⟨Agent A, Agent B, Key servK, Number Ts⟩

∈ parts (spies evs);

Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ ∃ ST. Says A B ⟨ST, Crypt servK ⟨Agent A, Number T3⟩⟩ ∈ set evs"

by (blast dest: servTicket\_authentic\_Tgs intro: Says\_K5)

text«The second assumption tells B what kind of key servK is.»

lemma B\_authenticates\_A\_r:

"[[ Crypt servK ⟨Agent A, Number T3⟩ ∈ parts (spies evs);

Crypt (shrK B) ⟨Agent A, Agent B, Key servK, Number Ts⟩

$\in \text{parts}(\text{spies evs});$ $\text{Crypt authK } \llbracket \text{Key servK, Agent B, Number Ts} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\text{Crypt (shrK A)} \llbracket \text{Key authK, Agent Tgs, Number Ta} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\sim \text{expiredSK Ts evs}; \sim \text{expiredAK Ta evs};$ $B \neq \text{Tgs}; A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbV} \rrbracket$ $\Rightarrow \exists \text{ST. Says A B } \llbracket \text{ST, Crypt servK } \llbracket \text{Agent A, Number T3} \rrbracket \rrbracket \in \text{set evs}"$ by (blast intro: Says_K5 dest: Confidentiality_B servTicket_authentic_Tgs) text« $\langle u\_B \text{ authenticates } A \rangle$ would be the same as $\langle B \text{ authenticates } A \rangle$ because the servK confidentiality assumption is yet unrelaxed» lemma u_B_authenticates_A_r: " $\llbracket \text{Crypt servK } \llbracket \text{Agent A, Number T3} \rrbracket \in \text{parts}(\text{spies evs});$ $\text{Crypt (shrK B)} \llbracket \text{Agent A, Agent B, Key servK, Number Ts} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\sim \text{expiredSK Ts evs};$ $B \neq \text{Tgs}; A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbV} \rrbracket$ $\Rightarrow \exists \text{ST. Says A B } \llbracket \text{ST, Crypt servK } \llbracket \text{Agent A, Number T3} \rrbracket \rrbracket \in \text{set evs}"$ by (blast intro: Says_K5 dest: u_Confidentiality_B servTicket_authentic_Tgs) lemma A_authenticates_B: " $\llbracket \text{Crypt servK (Number T3)} \in \text{parts}(\text{spies evs});$ $\text{Crypt authK } \llbracket \text{Key servK, Agent B, Number Ts} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\text{Crypt (shrK A)} \llbracket \text{Key authK, Agent Tgs, Number Ta} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\text{Key authK} \notin \text{analz}(\text{spies evs}); \text{Key servK} \notin \text{analz}(\text{spies evs});$ $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbV} \rrbracket$ $\Rightarrow \text{Says B A (Crypt servK (Number T3))} \in \text{set evs}"$ by (metis authK_authentic Oops_range_spies1 Says_K6 servK_authentic u_K4_imp_K2 unique_authKeys) lemma A_authenticates_B_r: " $\llbracket \text{Crypt servK (Number T3)} \in \text{parts}(\text{spies evs});$ $\text{Crypt authK } \llbracket \text{Key servK, Agent B, Number Ts} \rrbracket$ $\in \text{parts}(\text{spies evs});$ $\text{Crypt (shrK A)} \llbracket \text{Key authK, Agent Tgs, Number Ta} \rrbracket$	$\in \text{parts}(\text{spies evs});$ $\sim \text{expiredAK Ta evs}; \sim \text{expiredSK Ts evs};$ $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbV} \rrbracket$ $\Rightarrow \text{Says B A (Crypt servK (Number T3))} \in \text{set evs}"$ apply (frule authK_authentic) apply (erule_tac [3] exE) apply (frule_tac [3] Says_Kas_message_form) apply (frule_tac [4] Confidentiality_Kas) apply (frule_tac [7] servK_authentic) apply auto apply (metis Confidentiality_Tgs K4_imp_K2 Says_K6 unique_authKeys) done subsection«Parties' knowledge of session keys.  An agent knows a session key if he used it to issue a cipher. These  guarantees can be interpreted both in terms of key distribution  and of non-injective agreement on the session key.» lemma Kas_Issues_A: " $\llbracket \text{Says Kas A } \llbracket \text{Crypt (shrK A)} \llbracket \text{Key authK, Peer, Ta} \rrbracket, \text{authTicket} \rrbracket \in \text{set evs};$ $\text{evs} \in \text{kerbV} \rrbracket$ $\Rightarrow \text{Kas Issues A with (Crypt (shrK A)} \llbracket \text{Key authK, Peer, Ta} \rrbracket)$  on evs" apply (simp (no_asm) add: Issues_def) apply (rule exI) apply (rule conjI, assumption) apply (simp (no_asm)) apply (erule rev_mp) apply (erule kerbV.induct) apply (frule_tac [5] Says_ticket_parts) apply (frule_tac [7] Says_ticket_parts) apply (simp_all (no_asm_simp) add: all_conj_distrib) txt«K2» apply (simp add: takeWhile_tail)  apply (metis MPair_parts parts.Body parts_idem parts_spies_takeWhile_mono parts_trans spies_evs_rev usedI) done
---	--

lemma A\_authenticates\_and\_keydist\_to\_Kas:

"[[ Crypt (shrK A) {Key authK, Peer, Ta} ∈ parts (spies evs);

A ∉ bad; evs ∈ kerbV ]]

⇒ Kas Issues A with (Crypt (shrK A) {Key authK, Peer, Ta})

on evs"

by (blast dest!: authK\_authentic Kas\_Issues\_A)

lemma Tgs\_Issues\_A:

"[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}, servTicket}

∈ set evs;

Key authK ∉ analz (spies evs); evs ∈ kerbV ]]

⇒ Tgs Issues A with

(Crypt authK {Key servK, Agent B, Number Ts}) on evs"

apply (simp (no\_asm) add: Issues\_def)

apply (rule exI)

apply (rule conjI, assumption)

apply (simp (no\_asm))

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (frule\_tac [5] Says\_ticket\_parts)

apply (frule\_tac [7] Says\_ticket\_parts)

apply (simp\_all (no\_asm\_simp) add: all\_conj\_distrib)

apply (simp add: takeWhile\_tail)

(\*Last two thms installed only to derive authK ∉ range shrK\*)

apply (blast dest: servK\_authentic parts\_spies\_takeWhile\_mono [THEN subsetD])

parts\_spies\_evs\_revD2 [THEN subsetD] authTicket\_authentic

Says\_Kas\_message\_form)

done

lemma A\_authenticates\_and\_keydist\_to\_Tgs:

"[[ Crypt authK {Key servK, Agent B, Number Ts}

∈ parts (spies evs);

Key authK ∉ analz (spies evs); B ≠ Tgs; evs ∈ kerbV ]]

⇒ ∃A. Tgs Issues A with

(Crypt authK {Key servK, Agent B, Number Ts}) on evs"

by (blast dest: Tgs\_Issues\_A servK\_authentic\_bis)

lemma B\_Issues\_A:

"[[ Says B A (Crypt servK (Number T3)) ∈ set evs;

Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ B Issues A with (Crypt servK (Number T3)) on evs"

apply (simp (no\_asm) add: Issues\_def)

apply (rule exI)

apply (rule conjI, assumption)

apply (simp (no\_asm))

apply (erule rev\_mp)

apply (erule rev\_mp)

apply (erule kerbV.induct, analz\_mono\_contra)

apply (simp\_all (no\_asm\_simp) add: all\_conj\_distrib)

apply blast

txt:K6 requires numerous lemmas>

apply (simp add: takeWhile\_tail)

apply (blast intro: Says\_K6 dest: servTicket\_authentic

parts\_spies\_takeWhile\_mono [THEN subsetD])

parts\_spies\_evs\_revD2 [THEN subsetD])

done

lemma A\_authenticates\_and\_keydist\_to\_B:

"[[ Crypt servK (Number T3) ∈ parts (spies evs);

Crypt authK {Key servK, Agent B, Number Ts}

∈ parts (spies evs);

Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}

∈ parts (spies evs);

Key authK ∉ analz (spies evs); Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]]

⇒ B Issues A with (Crypt servK (Number T3)) on evs"

by (blast dest!: A\_authenticates\_B B\_Issues\_A)

(\*Must use ≤ rather than =, otherwise it cannot be proved inductively!\*)

(\*This is too strong for version V but would hold for version IV if only B

in K6 said a fresh timestamp.

lemma honest\_never\_says\_newer\_timestamp:

"[[ (CT evs) ≤ T ; Number T ∈ parts {X}; evs ∈ kerbV ]]	Key servK ∉ analz (spies evs);
⇒ ∀ A B. A ≠ Spy → Says A B X ∉ set evs"	B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
apply (erule rev_mp)	⇒ A Issues B with (Crypt servK {Agent A, Number T3}) on evs"
apply (erule kerbV.induct)	apply (simp (no_asm) add: Issues_def)
apply (simp_all)	apply (rule exI)
apply force	apply (rule conjI, assumption)
apply force	apply (simp (no_asm))
txt{*clarifying case K3*}	apply (erule rev_mp)
apply (rule impl)	apply (erule rev_mp)
apply (rule impl)	apply (erule kerbV.induct, analz_mono_contra)
apply (frule Suc_leD)	apply (frule_tac [7] Says_ticket_parts)
apply (clarify)	apply (frule_tac [5] Says_ticket_parts)
txt{*cannot solve K3 or K5 because the spy might send CT evs as authTicket or servTicket, which the honest agent would forward*}	apply (simp_all (no_asm_simp))
prefer 2 apply force	txt<K5>
prefer 4 apply force	apply auto
prefer 4 apply force	apply (simp add: takeWhile_tail)
txt{*cannot solve K6 unless B updates the timestamp - rather than bouncing T3*}	txt<Level 15: case study necessary because the assumption doesn't state the form of servTicket. The guarantee becomes stronger.>
oops	prefer 2 apply (simp add: takeWhile_tail)
*)	(**This single command of version IV...
text<But can prove a less general fact concerning only authenticators!>	apply (blast dest: Says_imp_spies [THEN analz.Inj, THEN analz_Decrypt']
lemma honest_never_says_newer_timestamp_in_auth:	K3_imp_K2 K4_trustworthy'
"[[ (CT evs) ≤ T; Number T ∈ parts {X}; A ∉ bad; evs ∈ kerbV ]]	parts_spies_takeWhile_mono [THEN subsetD]
⇒ Says A B {Y, X} ∉ set evs"	parts_spies_evs_revD2 [THEN subsetD]
apply (erule rev_mp)	intro: Says_Auth)
apply (erule kerbV.induct)	...expands as follows - including extra exE because of new form of lemmas*)
apply auto	apply (frule K3_imp_K2, assumption, assumption, erule exE, erule exE)
done	apply (case_tac "Key authK ∈ analz (spies evs5)")
lemma honest_never_says_current_timestamp_in_auth:	apply (metis Says_imp_analz_Spy analz.Fst analz_Decrypt')
"[[ (CT evs) = T; Number T ∈ parts {X}; A ∉ bad; evs ∈ kerbV ]]	apply (frule K3_imp_K2, assumption, assumption, erule exE, erule exE)
⇒ Says A B {Y, X} ∉ set evs"	apply (drule Says_imp_knows_Spy [THEN parts.Inj, THEN parts.Fst])
by (metis honest_never_says_newer_timestamp_in_auth le_refl)	apply (frule servK_authentic_ter, blast, assumption+)
lemma A_Issues_B:	apply (drule parts_spies_takeWhile_mono [THEN subsetD])
"[[ Says A B {ST, Crypt servK {Agent A, Number T3}} ∈ set evs;	apply (drule parts_spies_evs_revD2 [THEN subsetD])

txt«@{term Says\_K5} closes the proof in version IV because it is clear which

servTicket an authenticator appears with in msg 5. In version V an authenticator can appear with any item that the spy could replace the servTicket with»

apply (frule Says\_K5, blast)

txt«We need to state that an honest agent wouldn't send the wrong timestamp

within an authenticator, whatever it is paired with»

apply (auto simp add: honest\_never\_says\_current\_timestamp\_in\_auth)

done

lemma B\_authenticates\_and\_keydist\_to\_A:

"[[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);

Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}

∈ parts (spies evs);

Key servK ∉ analz (spies evs);

B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbV ]]

⇒ A Issues B with (Crypt servK {Agent A, Number T3}) on evs"

by (blast dest: B\_authenticates\_A A\_Issues\_B)

subsection«

Novel guarantees, never studied before. Because honest agents always say

the right timestamp in authenticators, we can prove unicity guarantees based

exactly on timestamps. Classical unicity guarantees are based on nonces.

Of course assuming the agent to be different from the Spy, rather than not in

bad, would suffice below. Similar guarantees must also hold of

Kerberos IV.»

text«Notice that an honest agent can send the same timestamp on two

different traces of the same length, but not on the same trace!»

lemma unique\_timestamp\_authenticator1:

"[[ Says A Kas {Agent A, Agent Tgs, Number T1} ∈ set evs;

Says A Kas' {Agent A, Agent Tgs', Number T1} ∈ set evs;

A ∉ bad; evs ∈ kerbV ]]

⇒ Kas=Kas' ∧ Tgs=Tgs'"

apply (erule rev\_mp, erule rev\_mp)

apply (erule kerbV.induct)

apply (auto simp add: honest\_never\_says\_current\_timestamp\_in\_auth)

done

lemma unique\_timestamp\_authenticator2:

"[[ Says A Tgs {AT, Crypt AK {Agent A, Number T2}, Agent B} ∈ set evs;

Says A Tgs' {AT', Crypt AK' {Agent A, Number T2}, Agent B'} ∈ set evs;

A ∉ bad; evs ∈ kerbV ]]

⇒ Tgs=Tgs' ∧ AT=AT' ∧ AK=AK' ∧ B=B'"

apply (erule rev\_mp, erule rev\_mp)

apply (erule kerbV.induct)

apply (auto simp add: honest\_never\_says\_current\_timestamp\_in\_auth)

done

lemma unique\_timestamp\_authenticator3:

"[[ Says A B {ST, Crypt SK {Agent A, Number T}} ∈ set evs;

Says A B' {ST', Crypt SK' {Agent A, Number T}} ∈ set evs;

A ∉ bad; evs ∈ kerbV ]]

⇒ B=B' ∧ ST=ST' ∧ SK=SK'"

apply (erule rev\_mp, erule rev\_mp)

apply (erule kerbV.induct)

apply (auto simp add: honest\_never\_says\_current\_timestamp\_in\_auth)

done

text«The second part of the message is treated as an authenticator by the last

simplification step, even if it is not an authenticator!»

lemma unique\_timestamp\_authticket:

"[[ Says Kas A {X, Crypt (shrK Tgs) {Agent A, Agent Tgs, Key AK, T}} ∈ set evs;

Says Kas A' {X', Crypt (shrK Tgs') {Agent A', Agent Tgs', Key AK', T}} ∈ set evs;

evs ∈ kerbV ]]

⇒ A=A' ∧ X=X' ∧ Tgs=Tgs' ∧ AK=AK'"

apply (erule rev\_mp, erule rev\_mp)

apply (erule kerbV.induct)

apply (auto simp add: honest\_never\_says\_current\_timestamp\_in\_auth)

done

text«The second part of the message is treated as an authenticator by the last

simplification step, even if it is not an authenticator!»

lemma unique\_timestamp\_servticket:

"[[ Says Tgs A {X, Crypt (shrK B) {Agent A, Agent B, Key SK, T}} ∈ set evs;

```

    Says Tgs A' {X', Crypt (shrK B')} {Agent A', Agent B', Key SK', T}} ∈ set evs;

    evs ∈ kerbV ]]

⇒ A=A' ∧ X=X' ∧ B=B' ∧ SK=SK'"

apply (erule rev_mp, erule rev_mp)

apply (erule kerbV.induct)

apply (auto simp add: honest_never_says_current_timestamp_in_auth)

done

(*Uses assumption K6's assumption that B ≠ Kas, otherwise B should say
fresh timestamp*)

lemma Kas_never_says_newer_timestamp:

  "[[ (CT evs) ≤ T; Number T ∈ parts {X}; evs ∈ kerbV ]]

  ⇒ ∀ A. Says Kas A X ∉ set evs"

apply (erule rev_mp)

apply (erule kerbV.induct, auto)

done

lemma Kas_never_says_current_timestamp:

  "[[ (CT evs) = T; Number T ∈ parts {X}; evs ∈ kerbV ]]

  ⇒ ∀ A. Says Kas A X ∉ set evs"

by (metis Kas_never_says_newer_timestamp eq_imp_le)

lemma unique_timestamp_msg2:

  "[[ Says Kas A {Crypt (shrK A) {Key AK, Agent Tgs, T}, AT} ∈ set evs;

  Says Kas A' {Crypt (shrK A') {Key AK', Agent Tgs', T}, AT'} ∈ set evs;

  evs ∈ kerbV ]]

  ⇒ A=A' ∧ AK=AK' ∧ Tgs=Tgs' ∧ AT=AT'"

apply (erule rev_mp, erule rev_mp)

apply (erule kerbV.induct)

apply (auto simp add: Kas_never_says_current_timestamp)

done

(*Uses assumption K6's assumption that B ≠ Tgs, otherwise B should say
fresh timestamp*)

lemma Tgs_never_says_newer_timestamp:

  "[[ (CT evs) ≤ T; Number T ∈ parts {X}; evs ∈ kerbV ]]

  ⇒ ∀ A. Says Tgs A X ∉ set evs"

apply (erule rev_mp)

apply (erule kerbV.induct, auto)

```

```

done

lemma Tgs_never_says_current_timestamp:

  "[[ (CT evs) = T; Number T ∈ parts {X}; evs ∈ kerbV ]]

  ⇒ ∀ A. Says Tgs A X ∉ set evs"

by (metis Tgs_never_says_newer_timestamp eq_imp_le)

lemma unique_timestamp_msg4:

  "[[ Says Tgs A {Crypt (shrK A) {Key SK, Agent B, T}, ST} ∈ set evs;

  Says Tgs A' {Crypt (shrK A') {Key SK', Agent B', T}, ST'} ∈ set evs;

  evs ∈ kerbV ]]

  ⇒ A=A' ∧ SK=SK' ∧ B=B' ∧ ST=ST'"

apply (erule rev_mp, erule rev_mp)

apply (erule kerbV.induct)

apply (auto simp add: Tgs_never_says_current_timestamp)

done

end

```

## ۶،۲ کد تحلیل پروتکل - Needham-

### Schröder با ابزار NPA-Maude

```

***(

The informal journal-level description of this protocol is as follows:

A --> B: pk(B,A ; N_A)

B --> A: pk(A, N_A ; N_B)

A --> B: pk(B, N_B)

where N_A and N_B are nonces, pk(x,y) means message y encrypted using
public

key x, and sk(x,y) means message y encrypted using private key x.

Moreover, encryption/decryption have the following algebraic properties:

pk(K,sk(K,M)) = M .

sk(K,pk(K,M)) = M .

)***

fmod PROTOCOL-EXAMPLE-SYMBOLS is

--- Importing sorts Msg, Fresh, Public, and GhostData

protecting DEFINITION-PROTOCOL-RULES .

-----

--- Overwrite this module with the syntax of your protocol

```

```

--- Notes:
protecting PROTOCOL-EXAMPLE-SYMBOLS .

--- * Sort Msg and Fresh are special and imported
protecting DEFINITION-PROTOCOL-RULES .

--- * Every sort must be a subsort of Msg
protecting DEFINITION-CONSTRAINTS-INPUT .

--- * No sort can be a superset of Msg
-----

--- Sort Information
-----
sorts Name Nonce Key .

subsort Name Nonce Key < Msg .

subsort Name < Key .

subsort Name < Public .

--- Encoding operators for public/private encryption

op pk : Key Msg -> Msg [frozen] .

op sk : Key Msg -> Msg [frozen] .

--- Nonce operator

op n : Name Fresh -> Nonce [frozen] .

--- Principals

op a : -> Name . --- Alice

op b : -> Name . --- Bob

op i : -> Name . --- Intruder

--- Associativity operator

op _;_ : Msg Msg -> Msg [gather (e E) frozen] .

endfm

fmod PROTOCOL-EXAMPLE-ALGEBRAIC is

protecting PROTOCOL-EXAMPLE-SYMBOLS .

-----

--- Overwrite this module with the algebraic properties

--- of your protocol
-----

var Z : Msg .

var Ke : Key .

*** Encryption/Decryption Cancellation

eq pk(Ke,sk(Ke,Z)) = Z [nonexec metadata "variant"] .

eq sk(Ke,pk(Ke,Z)) = Z [nonexec metadata "variant"] .

endfm

fmod PROTOCOL-SPECIFICATION is

protecting PROTOCOL-EXAMPLE-SYMBOLS .

protecting DEFINITION-PROTOCOL-RULES .

protecting DEFINITION-CONSTRAINTS-INPUT .

-----

--- Overwrite this module with the strands

--- of your protocol
-----

var Ke : Key .

vars X Y Z : Msg .

vars r r' : Fresh .

vars A B : Name .

vars N N1 N2 : Nonce .

eq STRANDS-DOLEVYAO

= :: nil :: [ nil | -(X), -(Y), +(X; Y), nil ] &

:: nil :: [ nil | -(X; Y), +(X), nil ] &

:: nil :: [ nil | -(X; Y), +(Y), nil ] &

:: nil :: [ nil | -(X), +(sk(i,X)), nil ] &

:: nil :: [ nil | -(X), +(pk(Ke,X)), nil ] &

:: nil :: [ nil | +(A), nil ]

[nonexec] .

eq STRANDS-PROTOCOL

= :: r ::

[ nil | +(pk(B,A; n(A,r))), -(pk(A,n(A,r); N)), +(pk(B, N)), nil ] &

:: r ::

[ nil | -(pk(B,A; N)), +(pk(A, N; n(B,r))), -(pk(B,n(B,r))), nil ]

[nonexec] .

eq ATTACK-STATE(0)

= :: r ::

[ nil, -(pk(b,a; N)), +(pk(a, N; n(b,r))), -(pk(b,n(b,r))) | nil ]

| | n(b,r) inl, empty

| | nil

| | nil

| | nil

[nonexec] .

eq ATTACK-STATE(1)

```



```

= :: r ::

[ nil, -(pk(b,a ; N)), +(pk(a, N ; n(b,r))), -(pk(b,n(b,r))) | nil ]

| | empty

| | nil

| | nil

| | never *** for authentication

(:: r' ::

[ nil, +(pk(b,a ; N)), -(pk(a, N ; n(b,r))) | +(pk(b,n(b,r))), nil ]

& S:StrandSet

| | K:IntruderKnowledge

[nonexec] .

endfm

--- THIS HAS TO BE THE LAST LOADED MODULE !!!!

select MAUDE-NPA .

```

## ۶،۳ نمونه حمله یافت شده توسط

### ابزار Maude-NPA

```

reduce in MAUDE-NPA : initials(0,unbounded) .

rewrites: 35 in 0ms cpu (0ms real) (~ rewrites/second)

result IdSystem: < 1 . 5 . 2 . 5 . 2 . 3 . 3 . 1 > (

:: nil ::

[ nil |

-(pk(i, n(b, #1:Fresh))),

+(n(b, #1:Fresh)), nil] &

:: nil ::

[ nil |

-(pk(i, a ; n(a, #0:Fresh))),

+(a ; n(a, #0:Fresh)), nil] &

:: nil ::

[ nil |

-(n(b, #1:Fresh)),

+(pk(b, n(b, #1:Fresh))), nil] &

:: nil ::

[ nil |

```

```

-(a ; n(a, #0:Fresh)),

+(pk(b, a ; n(a, #0:Fresh))), nil] &

:: #0:Fresh ::

[ nil |

+(pk(i, a ; n(a, #0:Fresh))),

-(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),

+(pk(i, n(b, #1:Fresh))), nil] &

:: #1:Fresh ::

[ nil |

-(pk(b, a ; n(a, #0:Fresh))),

+(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),

-(pk(b, n(b, #1:Fresh))), nil] )

| |

pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh)) !inl,

pk(b, n(b, #1:Fresh)) !inl,

pk(b, a ; n(a, #0:Fresh)) !inl,

pk(i, n(b, #1:Fresh)) !inl,

pk(i, a ; n(a, #0:Fresh)) !inl,

n(b, #1:Fresh) !inl,

(a ; n(a, #0:Fresh)) !inl

| |

+(pk(i, a ; n(a, #0:Fresh))),

-(pk(i, a ; n(a, #0:Fresh))),

+(a ; n(a, #0:Fresh)),

-(a ; n(a, #0:Fresh)),

+(pk(b, a ; n(a, #0:Fresh))),

-(pk(b, a ; n(a, #0:Fresh))),

+(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),

-(pk(a, n(a, #0:Fresh) ; n(b, #1:Fresh))),

+(pk(i, n(b, #1:Fresh))),

-(pk(i, n(b, #1:Fresh))),

+(n(b, #1:Fresh)),

-(n(b, #1:Fresh)),

+(pk(b, n(b, #1:Fresh))),

-(pk(b, n(b, #1:Fresh)))

```

||

nil

||

Nil

## ۶،۴ پروتکل Needham-Schruder به

### زبان Maude-PSL

/\*

The NSPK protocol is one of the default example protocols. It's nice and simple,

and it has a well-known attack that actually wasn't discovered for several years.

Therefore, it works wonderfully as an example both of what a protocol is, and of

the tremendous subtlety involved in crafting a secure protocol, and in verifying

that it is secure.

Andrew Cholewa

11/15/2013

Updated with comments: 12/2/2013

\*/

spec NSPK is

/\*

In the Equational Theory section, we define the language of the protocol,

i.e. the types, operators, as well as the equations that we need to verify the

protocol with respect to.

It has the exact same syntax as maude, except that we include the keywords type(s)

and subtype(s) as

synonyms for the keywords sort(s), and subsort(s) since this language is targeted

as much at protocol designers as it is formal specification people, and sort is

a rather odd word for what the rest of the computer science community refers to as

types.

\*/

Theory

types Name Nonce Enc .

subtype Name < Public .

op pk : Name Msg -> Enc .

op sk : Name Msg -> Enc .

op n : Name Fresh -> Nonce .

op \_ : Msg Msg -> Msg [ctor gather(e E)] .

ops a b i : -> Name [ctor] .

var X : Name .

var Z : Msg .

eq pk(X, sk(X,Z)) = Z .

eq sk(X, pk(X,Z)) = Z .

/\*

The Specification section is where we define the protocol message passing sequence,

the intruder capabilities, and define the attacks.

\*/

Protocol

vars AName BName : Name .

vars N1 N2 : Nonce .

vars r1 r2 : Fresh .

/\*

Input defines the input that each role requires in order to execute the protocol. For example, some protocols require each role to already have a

shared key with a trusted third party. Such a shared key would be the input

of the protocol. In this simple case, the only input are the people playing the actual roles.

\*/

roles A B .

In(A) = AName, BName .

In(B) = AName, BName .

/\*

Definition allows the user to define shorthand for certain commonly used

terms. This allows for two things:

1. It allows the specifier to abstract away from certain

implementation details (such as how nonces are specified)

2. It allows you to "hide" complicated terms behind a Name of some kind.

This way, whenever you wish to use that complicated term you only need

to use the Name, and if you need to modify the Name, you only need to

modify it in one place.

Note that the order in which the input and definition statements are

written do not matter.

\*/

Def(A) = na := n(AName, r1) .//na := na1 , na1 := na, na2 := pk(BName, sk(AName, na1)) . //na := n(AName, r1) .

Def(B) = nb := n(BName, r2) .

1 . A -> B : pk(BName, AName ; na) |- pk(BName, AName ; N1) .

2 . B -> A : pk(AName, N1 ; nb) |- pk(AName, na ; N2) .

3 . A -> B : pk(BName, N2) |- pk(BName, nb) .

Out(A) = na, N2 .

Out(B) = nb, N1 .

/\*

The intruder is where we specify the capabilities of the intruder.

The meaning of T1, T2, ..., Tn => T1', T2', ..., Tm' is:

If the intruder has the terms T1, T2, ..., Tn, he can derive the terms

T1', T2', ..., Tm'

T1, T2, ..., Tn <=> T1', T2', ..., Tm' . is equivalent to the two statements:

T1, T2, ..., Tn => T1', T2', ..., Tm' .

T1', T2', ..., Tm' => T1, T2, ..., Tn .

\*/

Intruder

vars X Y : Msg .

var A : Name .

var r1 : Fresh .

=> n(i, r1) .

X ; Y <=> X, Y .

X => sk(i, X) .

X, A => pk(A, X) .

/\*

Here we specify the attack states that we're interested in.

\*/

Attacks

0 .

/\*This allows us to take the generic roles specified in the message

passing

sequence and tie them to specific principals. Furthermore, we can

instantiate any variables in the terms belonging to B

In this example, we are saying that A and B are two different

people, and neither is the intruder.

\*/

Subst(B) = AName |-> a, BName |-> b .

//This tells us whose point of view we're looking at, and how much of

//the protocol the principal believes to have executed. Here, we are

//saying that principal B executes the entire principal.

B executes protocol .

//The knowledge that the intruder learns as a result of the protocol

//execution. Typically, you write here the knowledge you'd like to

//keep secret from the intruder.

Intruder learns nb .

1 .

B executes protocol .

Subst(B) = AName |-> a, BName |-> b .

Intruder learns nb .

/\*

Without statements say that we want to see if it's possible for

the statements above to occur without the statements below occurring.

The block following the without: statement is exactly the same as

the block above, except we don't allow Intruder learns statements.

What this attack asks, is whether it is possible for B to completely

execute his half of the protocol without A executing her half of

the protocol (and the intruder learns B's nonce to boot). Basically,

this asks if it's possible for B to be tricked into thinking he's

successfully spoken to Alice when in fact Alice hasn't been involved

at all.

\*/

without:

A executes protocol .

$\text{Subst}(A) = AName \mid \rightarrow a, BName \mid \rightarrow b .$

ends