Casper를 이용한 키 분배 프로토콜 명세

이성민, 김태윤
고려대학교 컴퓨터학과

A Key Distribution Protocol Specification using Casper

Sung-Min Lee, Tai-Yun Kim
Dept. of Computer Science & Engineering, Korea Univ.

요 약

최근 보안 프로토콜의 안전성을 검증하는 연구가 활발히 진행되고 있다. 이러한 연구 중 대표적인 방법이 CSP[1]와 FDR[2]과 같은 정형 기법을 이용하여 프로토콜을 분석하는 것이다. 이 방법은 매우 성공적으로 프로토콜의 문제점을 발견하고 안전성을 검증할 수 있었다. 하지만 CSP를 이용하여 프로토콜을 명세하는 것은 숙달된 전문가에게만 적합하고 시간이 오래 걸린다는 문제점을 갖는다. 이러한 문제점을 해결하여 좀 더 쉽게 프로토콜을 명세하여 CSP 코드를 생성해주는 캐스퍼(Casper)[3]가 제안되었다.

본 논문에서는 세 개의 메시지 교환을 통해서 키를 분배하는 키 분배 프로토콜을 제안하고 캐스퍼를 이용하여 제안한 프로토콜에 대한 정형 명세를 한다. 명세된 스크립트는 캐스퍼를 이용하여 FDR의 입력 파일이 되는 667 라인의 CSP 코드를 생성하였다. 캐스퍼를 이용한 정형 명세는 좀 더 편리하게 프로토콜을 명세할 수 있고 명세 시 발생할 수 있는 에러를 줄임으로써 신뢰성을 높일 수 있다.

1. 서 론

오늘날 통신 기술의 발달로 많은 정보들이 네트워크를 통해서 송수신 된다. 통신망을 이용한 정보의 교환 시 데이터의 무결성 및 개인의 프라이버시 보장이 매우 중요하다. 이러한 요구 때문에 데이터 송수신시 암호화/복호화 과정을 거치며 통신하는 방법이 많이 이용된다. 암호화/복호화는 통신 주체들 사이에 키 교환을 필요로 한다. 키를 교환하게 하는 프로토콜을 키 분배 프로토콜이라고 하는데[6], 키 분배와 같은 보안 프로토콜의 안전성 검증은 매우 중요한 문제이다.

최근에 보안 프로토콜의 안전성을 검증하는 연구가 활발히 진행되고 있는데, 정형 기법을 이용한 방법이 각광을 받고 있다. 대표적인 기법으로 프로세스 대수인 CSP를 이용하여 보안 프로토콜을 명세하고 그것을 모델 검사기(model checker)인 FDR을 이용하여 안전하지 않은 상태(state)가 존재하는지 추적하는 것이다. 이러한 추적을 통해서 보안 프로토콜 내의 홀(hole)을 발견할 수 있다[4,5]. 하지만 한 프로토콜을 CSP를 이용하여 명세하는 것은 매우 시간이 오래걸리고 CSP 문법에 익숙한 전문가에게만 적합한 일이다. 이러한 문제 해결을 위해서 Lowe에 의해서 제안된 방법이 캐스퍼(Casper)이다. 캐스퍼를 이용하면 CSP를 이용할 때에 비해서 좀 더 쉽게 프로토콜을 명세할 수 있다.

본 논문에서는 키 분배 프로토콜을 제안하고 그 프로토콜을 캐스퍼를 이용하여 명세한다. 캐스퍼 입력 파일은 9개의 섹션으로 구성되기 때문에 각각의 섹션에 대하여 제안한 프로토콜을 명세하여 CSP 코드를 생성하였다.

본 논문의 구성은 다음과 같다. 제 2장에서 CSP와 FDR을 이용하여 프로토콜의 안전성을 분석하는 기법과 캐스퍼에 대하여 논하고, 제 3장에서는 제안한 프로토콜을 제시하고 캐스퍼를 이용하여 명세한다. 마지막으로 4장에서 본 논문을 결론 맺고 향후 연구 과제를 제시한다.

2. 관련 연구

본 장에서는 정형 기법을 이용하여 보안 프로토콜을 명세 및 검증하는 기법을 살펴보고 캐스퍼에 대해서 간단히 설명한다.

2.1 정형 기법을 이용한 보안 프로토콜 안전성 분석

보안 프로토콜에서의 안전성 분석은 매우 중요하다. 이러한 분석 없이 프로토콜을 사용하게되면 중요한 정보가 유출되는 등의 사고가 발생할 수 있다. 이러한 문제를 해결하는 방법 중 하나가 프로토콜을 정형 명세하고 그것을 모델 검사기(model checker)를 이용하여 검증하는 것이다.

현재 보안 프로토콜을 명세하고 검증하는 다수의 정형 기법들이 존재한다. 이들 중에 대표적으로 많이 사용되는 기법으로 Hoare가

만든 프로세스 대수(process algebra)인 CSP(Communicating Sequential Processes)를 이용하여 보안 프로토콜에 대해서 명세하는 방법이 있다. 이렇게 명세된 CSP 코드는 Formal Systems에서 만든 모델 검사기인 FDR을 이용하여 안전성을 검증한다. FDR은 가능한 모든 상태(state)들을 방문하면서 안전하지 않은 상태가 존재하는지를 자동으로 검사한다. 이러한 기법은 매우 성공적이라는 평가를 받았고 CSP와 FDR을 이용해서 보안 프로토콜을 검증하는 절차는 다음과 같이 나타낼 수 있다[4,5].

- 프로토콜에 참여하는 모든 에이전트를 CSP 프로세스로 명세한다.
- 프로토콜과 상호작용할 수 있는 침입자를 CSP 프로세스로 명세한다.
- 명세한 프로토콜에 대해서 FDR을 이용하여 안전성을 검증한다.

2.2 Casper

CSP와 FDR을 이용한 방법은 매우 성공적임에도 불구하고 몇 가지 문제점을 갖는다. CSP를 이용하여 시스템을 명세하는 것은 매우 시간이 오래 걸리고 CSP 문법에 익숙한 전문가만이 할 수 있다. 또한 이러한 전문가 조차도 명세시 실수할 수 있다는 점이다. 이러한 문제를 해결한 것이 Lowe가 개발한 캐스퍼(Casper)이다[3].

캐스퍼는 함수형 언어(funcational language)인 Haskell로 쓰여진 컴파일러이다. 캐스퍼 스크립트는 CSP에 비해서 매우 간단하고 추상적으로 프로토콜을 표현한다. 이러한 스크립트를 이용하여 CSP 코드를 생성한다. 캐스퍼 스크립트는 다음과 같이 9개의 섹션을 이용하여 프로토콜을 명세한다.

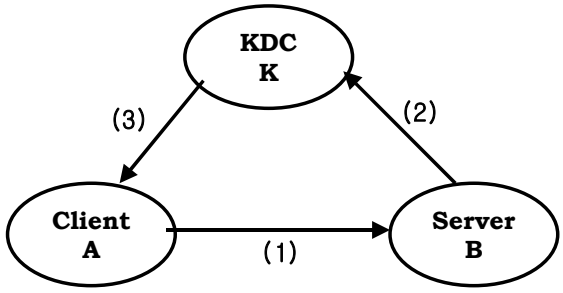
script := free variables section
processes section
protocol description section
specification section
[equivalences section]
actual variable section
[functions section]
system section
intruder section

3. Casper를 이용한 제안한 프로토콜의 명세

본 장에서는 키 분배 프로토콜을 제안하고 캐스퍼를 이용하여 프로토콜을 명세한 후 CSP 코드를 생성하는 기법을 기술한다.

3.1 제안한 키 분배 프로토콜

제안한 프로토콜은 클라이언트 A와 서버 B 사이에서 세션키 rB를 교환하는 키분배 프로토콜이다. 키교환 절차는 그림 1과 같다. 메시지 1에서 A는 KDC의 공개키로 자신이 생성한 키 암호화키 rA와 신분을 암호화하고 이러한 메시지에 자신의 비밀키로 서명한 값을 B에게 전달한다. 그러면 B는 A의 공개키를 가지고 받은 메시지를 복호화한 후 세션키 rB를 생성하고 메시지2와 같은 형식의 메시지를 만들어서 K에게 전송한다. K는 전달 받은 메시지를 복호화하고 Vernam 사이



Message 1 : $S_A(P_K(I_B,r_A),T_A)$
Message 2 : $S_B(I_A, P_K(I_B,r_A), P_K(I_A,r_B), T_B), Er_B(T_A,L)$
Message 3 : $I_B, V(r_A,r_B), T_K, S_K(I_B, V(r_A,r_B), T_K), Er_B(T_A,L)$

그림 1 키 교환 단계

퍼를 이용하여 rA와 rB를 암호화하고 타임스탬프 정보와 함께 A에게 전송한다. 그러면 A는 자신이 생성했던 키 rA를 이용하여 rB를 얻는다. 따라서 A와 B는 세션키 rB를 공유할 수 있다.

3.2 프로토콜 명세

그림 1과 같은 프로토콜을 캐스퍼를 통해서 명세하면 다음과 같다. free variable section에는 프로토콜에 참여할 에이전트들과 사용되는 함수 및 변수를 선언한다. PK, SPK는 공개키를 얻기 위한 함수이고 SK와 SSK는 비밀키를 얻기 위한 함수이다.

```
#Free variables
a,b : Agent
s : Server
ka, kb : SessionKey
PK : Agent -> PublicKey
SK : Agent -> SecretKey
SPK : Server -> ServerPublicKey
SSK : Server -> ServerSecretKey
InverseKeys = (PK, SK), (SPK, SSK), (ka, ka), (kb, kb)
ta, tb, ts : Nonce
```

프로토콜 상에 있는 각 에이전트들은 CSP 프로세스로 표현된다. CSP 프로세스의 이름은 각 에이전트를 표현한다. knows라는 키워드를 이용하여 각 에이전트들이 알고 있는 초기 지식을 표현한다. 제안된 프로토콜에서의 A는 자신의 신분 a, KDC의 신분 s, 세션키 ka, 타임스템프 ta, 공개키 함수 PK, SPK, 자신의 비밀키 SK(a)를 초기 지식으로 알고 있다고 설정한다.

```
#Processes
INITIATOR(a,s,ka,ta) knows PK, SPK, SK(a)
RESPONDER(b,s,kb,tb) knows PK, SPK, SK(b)
SERVER(s,ts) knows PK, SSK(s)
```

프로토콜 정의 시 중요한 부분은 전송하는 메시지에 대한 순차적인 정의이다. {m}{k} 기호는 메시지 m을 키 k로 암호화하는 것을 의미하고 ka (+) kb는 ka와 kb의 Vernam 암호화를 의미한다. 또한

그림 1의 메시지 1과 2를 명세할때 %기호를 이용하는데, 이것은 {b,ka}{SPK(s)}를 b가 복호화하지 않고 s에게 바로 전송할 때 사용한다. 제안된 프로토콜은 아래와 같이 명세될 수 있다.

```
#Protocol description
0. -> a : b
[a!=b]
1. a -> b : ta, {{b, ka}{SPK(s)} % v, ta}{SK(a)}
[a!=b]
2. b -> s : a, tb, {a, v % {b, ka}{SPK(s)}, {a,kb}{SPK(s)}, tb}{SK(b)}, {ta}{kb}
[a!=b]
3. s -> a : b, ka (+) kb, ts, {b, ka (+) kb, ts}{SSK(s)}, {ta}{kb}
[a!=b]
```

현재 캐스퍼에서는 secrets과 authentication, 두 종류의 명세를 제공한다. secrets은 안전하게 전송되어야하는 데이터를 명시하고 agreement는 두 에이전트 사이에서 인증을 명세하기 위해 사용된다.

```
#Specification
Secret(a, ka, [b,s])
Secret(b, kb, [a,s])
Agreement(a,b,[ta])
Agreement(b,s,[tb])
Agreement(s,a,[ts,ta])
```

실제 사용될 변수의 정의를 위해서 actual variables 섹션에 명시한다. 제안된 프로토콜에서 사용될 변수는 다음과 같이 정의하였다.

```
#Actual variables
A, B, M : Agent
S : Server
rA, rB, rM : SessionKey
Ta, Tb, Ts, Tm: Nonce
InverseKeys = (rA,rA), (rB,rB), (rM,rM)
```

제안된 프로토콜에서 사용되는 공개키/비밀키 함수는 다음과 같이 정의된다.

```
#Functions
symbolic PK, SK, SPK, SSK
```

시스템 섹션에서는 다음과 같이 각 에이전트에 실제 사용할 변수를 명시해 준다.

```
#System
INITIATOR(A,S,rA,Ta)
RESPONDER(B,S,rB,Tb)
SERVER(S,Ts)
```

마지막으로 침입자의 행위는 침입자의 이름과 초기 지식을 명시함으로써 제안한 프로토콜의 명세를 마무리할 수 있다. 침입자는 키 교환을 원하는 A,B와 KDC S를 알고 있고 이들의 공개키와 자신의 비밀키 및 자신이 생성한 세션키와 타임스탬프를 초기 지식으로 한다.

```
#Intruder Information
Intruder = M
IntruderKnowledge = {A, B, S, M, PK, SPK, rM, Tm, SK(M)}
```

이러한 캐스퍼 입력 스크립트를 만들고 컴파일했을때 667라인의 CSP 코드가 생성되었다.

4. 결 론

본 논문에서는 통신 주체들 사이에서 키 분배 및 인증 서비스를 제공하는 프로토콜을 제안하고 정형 명세를 하였다. 최근 보안 프로토콜의 분석은 CSP와 FDR과 같은 정형 기법을 이용하여 진행되고 있고 프로토콜의 문제점 발견 및 안전성 검증에 성공적임이 증명되었다.

하지만 프로토콜을 CSP를 이용하여 명세하는 것은 시간이 오래 걸리고 CSP 문법에 익숙한 전문가에게만 적합하다. 따라서 본 논문에서는 제안한 프로토콜을 캐스퍼를 이용하여 정형 명세하였다. 명세된 스크립트는 캐스퍼를 이용하여 FDR의 입력 파일이 되는 667 라인의 CSP 코드를 생성하였다. 캐스퍼를 이용한 정형 명세는 좀 더 편리하게 프로토콜을 명세할 수 있고 명세 시 발생할 수 있는 에러를 줄여서 신뢰성을 높일 수 있다.

캐스퍼를 이용하여 명세된 스크립트를 검증 툴인 FDR을 사용하여 프로토콜을 검증하는 것이 향후 연구 과제이다.

참 고 문 헌

[1] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[2] Formal Systems Ltd. *Failures-Divergence Refinement-FDR 2 User Manual,* 1997. Available via URL http://www.formal.demon.co.uk/FDR2.html

[3] Gavin Lowe, "Casper: A compiler for the analyis of security protocols," 1996.

[4] Gavin Lowe, Bill Roscoe. "Using CSP to Detect Errors in the TMN Protocol," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 23, NO. 10, pp. 659-669, OCTOBER 1997.

[5] A. W. Roscoe and M. H. Goldsmith, "The perfect "spy" for model checking cryptoprotocols," In Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols, 1997.

[6] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *HANDBOOK OF APPLIED CRYPTOGRAPHY*, CRC PRESS, 1996.