安全协议设计与分析

第八讲：ProVerif介绍及案例

李晖 网络空间安全学院

上讲内容

1. 形式化方法概述
2. Dolev-Yao模型
3. 基于逻辑推理的分析方法
4. 基于模型检测的分析方法
5. 基于定理证明的分析方法
6. 密码学可证明安全性分析方法
7. 常用分析工具

本次课程

1. ProVerif简介
2. ProVerif架构
3. 应用π-演算
4. 编码及运行结果
5. 类型检查
6. Horn子句
7. FIDO UAF协议的形式化分析
8. 总结

1.ProVerif简介

- 基于Dolev-Yao模型的形式化自动验证密码学协议工具
- Bruno Blanchet 团队开发
- 用 **Prolog 语言**实现协议建模
- 它能够描述各种密码学原语
  - 包括：共享密钥密码学和公钥密钥密码学，Hash 函数和Deffie-Hellman密钥交换协议，并指定了重写规则和方程式，输入语言为**应用 π- 演算**或 **Horn 子句**。
- 能处理无穷会话并发协议和无穷消息空间，克服了状态空间爆炸的问题。
- 如果协议有漏洞，此工具可给出相应的攻击序列。

1.ProVerif简介

- **特征**:
  - **基于符号模型或基于Dolev-Yao（1983）模型**
    - 完美密码学假设（Perfect cryptography assumption）
    - 消息是基于密码学原语的术语（terms）
    - 攻击者可以完全控制网络
  - **协议建模**
    - 应用π-演算
  - **安全属性定义（specification）**
    - 迹属性（Trace properties）（**语义机密性**）
    - 等价属性（Equivalence properties）（**强机密性**）
  - **算法**
    - Horn clauses Abstraction
    - Resolution Horn Clauses by Unification

1.ProVerif简介

用Proverif验证的安全协议

- Email protocol
- JFK protocol (Just Fast Keying)一个在Ipsec中代表IKE作为密钥交换的协议
- certified mailing-list protocol (Khurana and Hahm, 2006)
- e-voting protocols (Kremer and Ryan,2005;
- The ad-hoc routing protocol (专用路由协议) (Godskesen, 2006)
- the Trusted Platform Module(TPM) (Delaune et al., 2011)
- e-auction protocols 电子拍卖(Dreier et al., 2013)
- zero-knowledge protocols (Backes et al., 2008b; Backes et al., 2015)
- FIDO UAF protocol(our work, 2021)

2.ProVerif架构

1)通过应用π-演算描述协议

2)描述协议安全属性

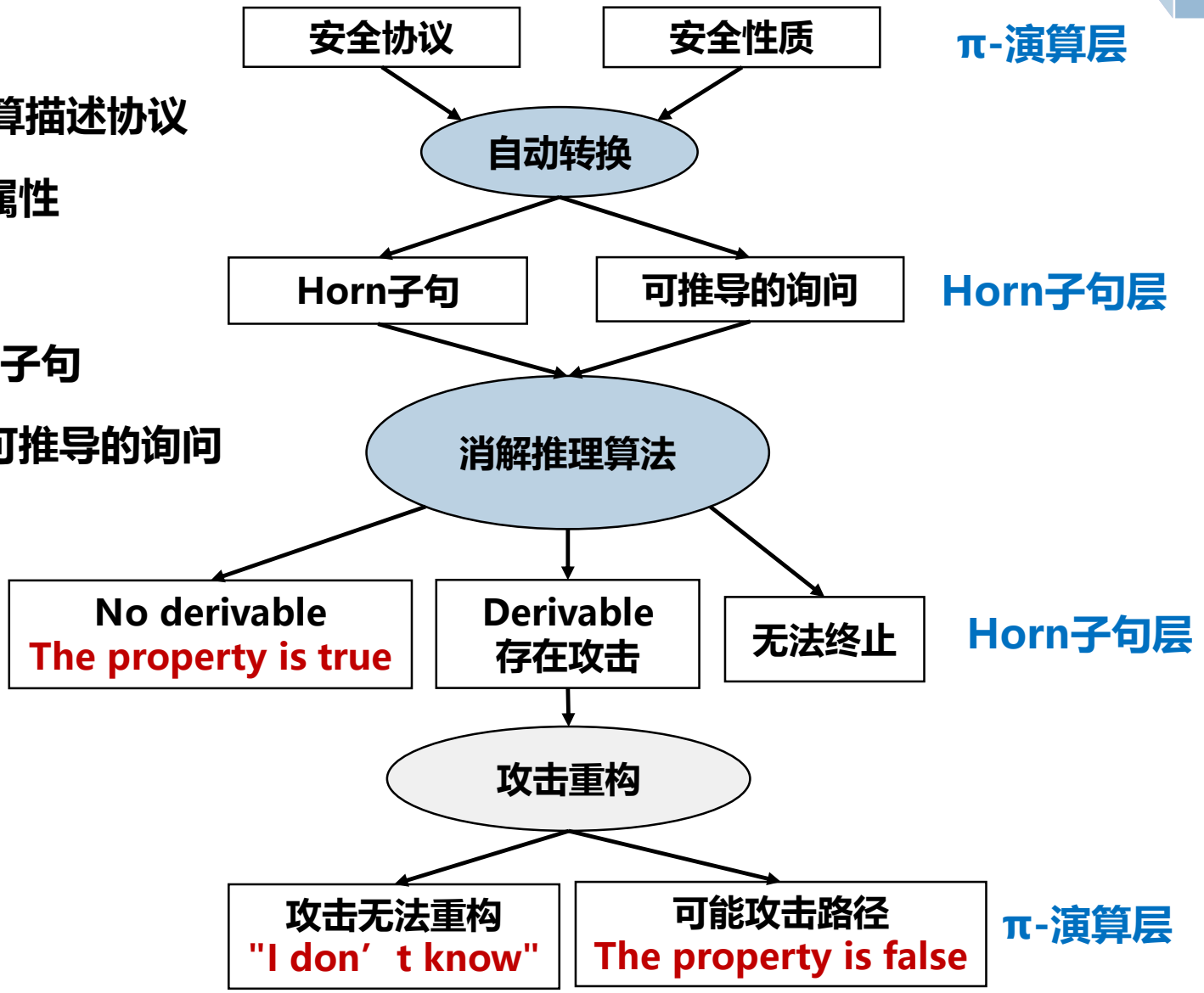
3)进行自动转换

4)协议转为Horn子句

5)安全性质转为可推导的询问

6)消解推理算法

7)攻击重构



3. π-演算基本语法

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, \ldots, M_n)$	constructor application
$D ::=$	expressions
M	term
$h(D_1, \ldots, D_n)$	function application
fail	failure
$P, Q ::=$	processes
$\mathbf{0}$	nil
$\mathsf{out}(N, M); P$	output
$\mathsf{in}(N, x : T); P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\mathsf{new}\ a : T; P$	restriction
$\mathsf{let}\ x : T = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$	expression evaluation
$\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional

3. π-演算的语法规则

Terms：术语

术语可以是数据，也可以是消息，术语是需要定义的，规则中的定义有：

- Channel：信道
- Bool：布尔值
- Bitstring：bit串

用户可以自定义术语的类型，但是在默认情况下，ProVerif在验证安全属性时忽略术语的类型，来让攻击者更有效地实现攻击。

3. π-演算的语法规则

expressions：表达式

- **构造函数**：**f(T1,T2……Tn)**
- **析构函数**：**g(T1,T2……Tn)**
- **其他函数**：**h(T1,T2……Tn)**

如

senc(M,K)是对称密钥加密函数，数据为M，对称密钥为K

sdec(senc(M,K),K) -> M 是senc对应的析构函数，通过密钥K还原M信息。

sign(M,skey)是签名函数，数据为M，私钥为skey

check(sign(M.skey),pkey) -> M 是sign对应的析构函数，通过公钥进行验证签名。

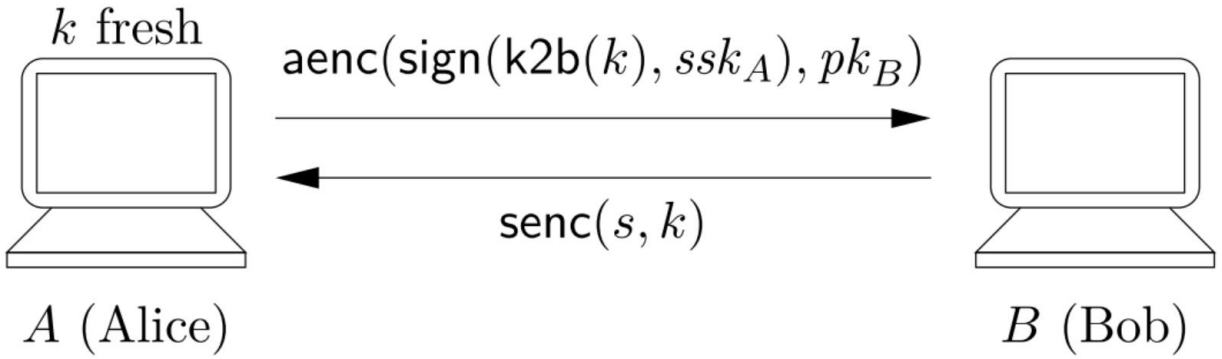
3. π-演算的语法规则

Process：进程

- **Out(c,x)** 向信道c中输出x
- **In(c,x:bitstring)** 从信道c中获取x
- **P|Q** 表示将P进程和Q进程平行执行
- **!P**表示P进行无限数量的执行
- **New a:key** 表示创建一个新的变量，可以是一个密钥，或一个随机数nonce
- **Let x:bitsting T= D in P else Q** 表示判断D是否为真，如果为真，x与M进行绑定，并执行P，否则执行Q
- **If M then P else Q** 表示判断M为真，执行P，否则执行Q

3. π-演算描述的一个简单协议

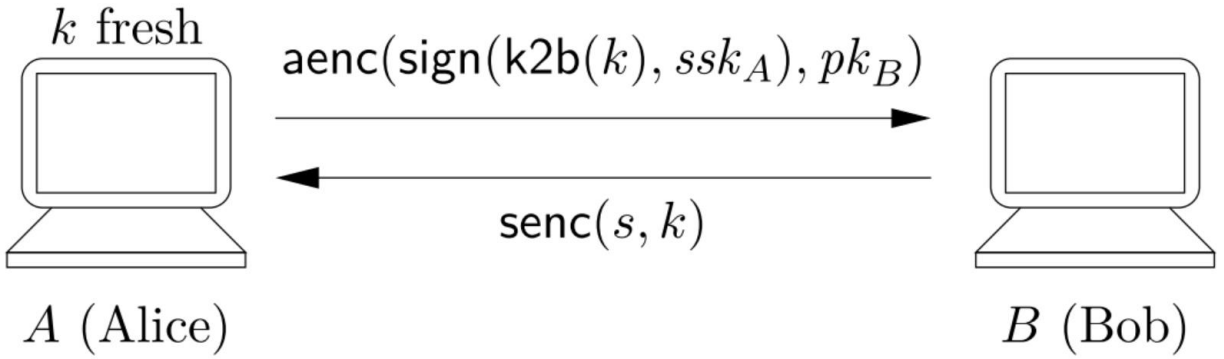
协议流程 V1



$$P_0 = \text{new } ssk_A : \text{skey}; \text{new } sk_B : \text{skey}; \text{let } spk_A = \text{pk}(ssk_A) \text{ in}$$
$$\text{let } pk_B = \text{pk}(sk_B) \text{ in } \text{out}(c, spk_A); \text{out}(c, pk_B);$$
$$(P_A(ssk_A, pk_B) \mid P_B(sk_B, spk_A))$$
$$P_A(ssk_A, pk_B) = !\, \text{new } k : \text{key};$$
$$\text{out}(c, \text{aenc}(\text{sign}(\text{k2b}(k), ssk_A), pk_B));$$
$$\text{in}(c, x : \text{bitstring}); \text{let } z = \text{sdec}(x, k) \text{ in } \mathbf{0}$$
$$P_B(sk_B, spk_A) = !\, \text{in}(c, y : \text{bitstring}); \text{let } y' = \text{adec}(y, sk_B) \text{ in}$$
$$\text{let } x_k = \text{b2k}(\text{check}(y', spk_A)) \text{ in } \text{out}(c, \text{senc}(s, x_k))$$

3. π-演算描述的一个简单协议

协议流程 V2



$$P_0 = \text{new } ssk_A : \text{skey}; \text{new } sk_B : \text{skey}; \text{let } spk_A = \text{pk}(ssk_A) \text{ in}$$
$$\text{let } pk_B = \text{pk}(sk_B) \text{ in } \text{out}(c, spk_A); \text{out}(c, pk_B);$$
$$(P_A(ssk_A, pk_B) \mid P_B(sk_B, spk_A))$$

$$P_A(ssk_A, pk_B) = !\,\text{in}(c, x_{pk_B} : \text{pkey}); \text{new } k : \text{key};$$
$$\text{out}(c, \text{aenc}(\text{sign}(\text{k2b}(k), ssk_A), x_{pk_B}));$$
$$\text{in}(c, x : \text{bitstring}); \text{let } z = \text{sdec}(x, k) \text{ in } \mathbf{0}$$

$$P_B(sk_B, spk_A) = !\,\text{in}(c, y : \text{bitstring}); \text{let } y' = \text{adec}(y, sk_B) \text{ in}$$
$$\text{let } x_k = \text{b2k}(\text{check}(y', spk_A)) \text{ in } \text{out}(c, \text{senc}(s, x_k))$$

3. π-演算描述的一个简单协议

安全目标（或安全属性）

- Specifying the properties:
  - **query not** *attacker(***s***).*

4. 编码及运行结果

free c: channel.

type key.
type pkey.
type skey.
type spkey.
type sskey.

fun k2b(key):bitstring $[data, typeConverter]$.
reduc forall k:key; b2k(k2b(k)) = k.

4.编码及运行结果

fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall x: bitstring, y: skey; adec(aenc(x, pk(y)), y) = x.

fun spk(sskey): spkey.
fun sign(bitstring, sskey): bitstring.
reduc forall m: bitstring, k: sskey; checksign(sign(m, k), spk(k)) = m.

fun senc(bitstring, key): bitstring.
reduc forall x: bitstring, y: key; sdec(senc(x, y), y) = x.

//Specification
free s: bitstring [*private*].
query *attacker*(s).

4.编码及运行结果

//PA will be revised in 2nd version

$$
\begin{aligned}
&\textbf{let } PA(sskA\text{: sskey}, pkB\text{:pkey}) = \\
&\qquad \textbf{new } k\text{:key}; \\
&\qquad \textbf{out}(\mathsf{c}, \mathsf{aenc}(\mathsf{sign}(\mathsf{k2b}(k), sskA), pkB)); \\
&\qquad \textbf{in}(\mathsf{c}, x\text{:bitstring}); \\
&\qquad \textbf{let } z = \mathsf{sdec}(x, k) \textbf{ in } 0.
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{let } PB(skB\text{:skey}, spkA\text{:spkey}) = \\
&\qquad \textbf{in}(\mathsf{c}, y\text{:bitstring}); \\
&\qquad \textbf{let } y1 = \mathsf{adec}(y, skB) \textbf{ in} \\
&\qquad \textbf{let } xk = \mathsf{b2k}(\mathsf{checksign}(y1, spkA)) \textbf{ in} \\
&\qquad \textbf{out}(\mathsf{c}, \mathsf{senc}(\mathsf{s}, xk)).
\end{aligned}
$$

4.编码及运行结果

$$
\begin{aligned}
&\textbf{process}\\
&\quad \textbf{new}\ sskA\text{: sskey};\\
&\quad \textbf{new}\ skB\text{: skey};\\
&\qquad \textbf{let}\ spkA = \mathsf{spk}(sskA)\ \textbf{in}\\
&\qquad \textbf{let}\ pkB = \mathsf{pk}(skB)\ \textbf{in}\\
&\qquad \textbf{out}(\mathsf{c}, spkA);\\
&\qquad \textbf{out}(\mathsf{c}, pkB);\\
&\qquad (!PA(sskA, pkB)\ |!PB(skB, spkA))
\end{aligned}
$$

4.编码及运行结果

```
// PA in 2nd version
let PA(sskA: sskey, pkB:pkey) =
        in(c, xpkB:pkey);
        new k:key;
        out(c, aenc(sign(k2b(k), sskA), xpkB));
        in(c, x:bitstring);
        let z = sdec(x, k) in 0.

// Recall PA in 1st version
let PA(sskA: sskey, pkB:pkey) =
        new k:key;
        out(c, aenc(sign(k2b(k), sskA), pkB));
        in(c, x:bitstring);
        let z = sdec(x, k) in 0.
```

4.编码及运行结果

//Results in 1st version:

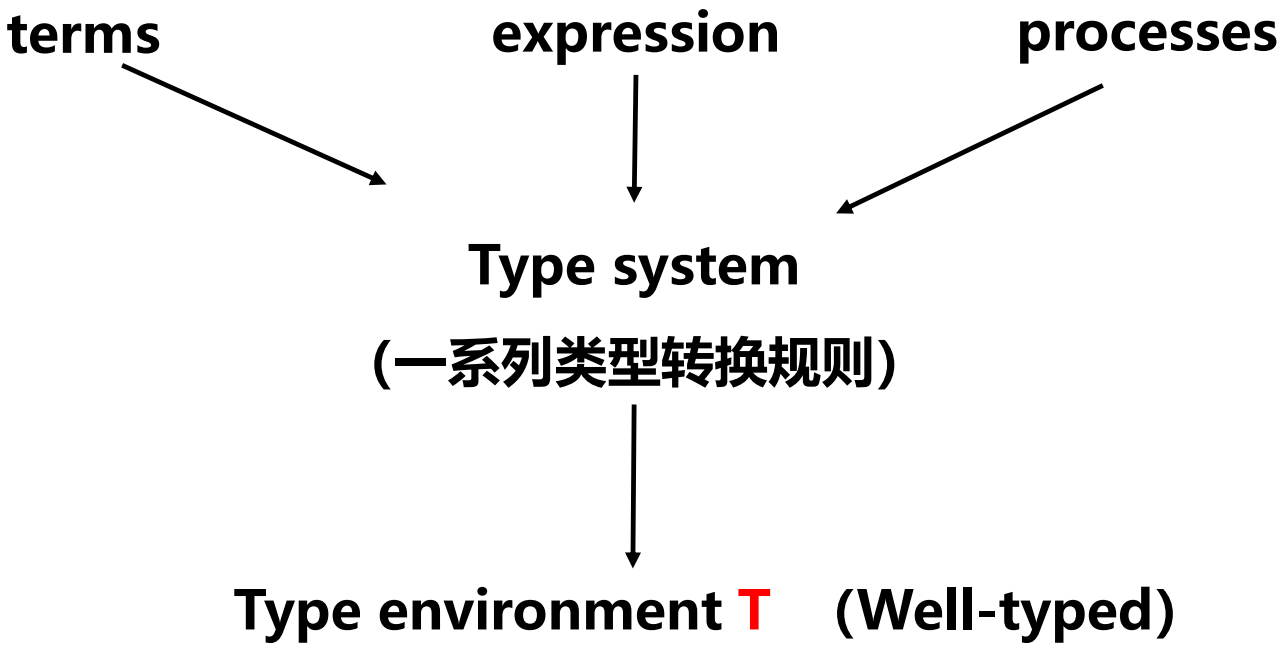
Query not attacker(s[]) is true.

//Results in 2st version:

Query not attacker(s[]) is false.

5. 类型检查 Type-System

目的：帮助用户检查编写协议的正确性。



6. Horn子句

1) Horn子句表示法

- 协议的Horn子句表示法使用的主要**谓词**是**攻击者**：

  attacker(*M*): 表示“攻击者可以获得M”

  message(p, p')：表示“信息P’ 出现在信道P上”

- 利用这些谓词，我们可以对**攻击者**和**协议参与者**的行为建模。

- 进程（Process）会**自动转换**为Horn子句

6. Horn子句

- **Model Dolev-Yao adversary in Horn clauses**
  - **The adversary can *overhear*, *send*, and *synthesize* any message**

- *Overhear*: $\mathbf{message}(x, y) \wedge \mathbf{attacker}(x) \Rightarrow \mathbf{attacker}(y)$
  - the adversary can listen on all channels it has
- *Send*: $\mathbf{attacker}(x) \wedge \mathbf{attacker}(y) \Rightarrow \mathbf{message}(x, y)$
  - it can send all messages it has on all channels it has
- *Synthesize*:
  - $\mathbf{attacker}(x_1) \wedge \cdots \wedge \mathbf{attacker}(x_n) \Rightarrow \mathbf{attacker}(f(x_1, \ldots, x_n))$
    - constructor f
  - $\mathbf{attacker}(U_1) \wedge \cdots \wedge \mathbf{attacker}(U_n) \Rightarrow \mathbf{attacker}(U)$
    - destructor $g(U_1, \ldots, U_n) = U$

6. Horn子句

- Constructors $f(M_1, \ldots, M_n)$
  $\text{attacker}(x_1) \wedge \ldots \wedge \text{attacker}(x_n) \rightarrow \text{attacker}(f(x_1, \ldots, x_n))$

Example: Shared-key encryption $\text{encrypt}(m, k)$

$\text{attacker}(m) \wedge \text{attacker}(k) \rightarrow \text{attacker}(\text{encrypt}(m, k))$

- Destructors $g(M_1, \ldots, M_n) \rightarrow M$
  $\text{attacker}(M_1) \wedge \ldots \wedge \text{attacker}(M_n) \rightarrow \text{attacker}(M)$

Example: Shared-key decryption $\text{decrypt}(\text{encrypt}(m, k), k) \rightarrow m$

$\text{attacker}(\text{encrypt}(m, k)) \wedge \text{attacker}(k) \rightarrow \text{attacker}(m)$

6. Horn子句

Examples for *Synthesize*:

- Constructors

$$\mathsf{attacker}(m) \wedge \mathsf{attacker}(k) \Rightarrow \mathsf{attacker}(\mathsf{senc}(m,k)) \quad (\mathsf{senc})$$

$$\mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(\mathsf{pk}(sk)) \quad (\mathsf{pk})$$

$$\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(m,pk)) \quad (\mathsf{aenc})$$

$$\mathsf{attacker}(m) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(\mathsf{sign}(m,sk)) \quad (\mathsf{sign})$$

- Destructors

$$\mathsf{attacker}(\mathsf{senc}(m,k)) \wedge \mathsf{attacker}(k) \Rightarrow \mathsf{attacker}(m) \quad (\mathsf{sdec})$$

$$\mathsf{attacker}(\mathsf{aenc}(m,\mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \Rightarrow \mathsf{attacker}(m) \quad (\mathsf{adec})$$

$$\mathsf{attacker}(\mathsf{sign}(m,sk)) \wedge \mathsf{attacker}(\mathsf{pk}(sk)) \Rightarrow \mathsf{attacker}(m) \quad (\mathsf{check})$$

6. Horn子句

2) 协议子句转换规则

$$[\![P]\!]\rho s H$$

- P：π演算的进程描述
- ρ：变量代替的方法，如a->a[i]
- s：当前的会话标识和输入的变量
- H：当前的假设H

$$[\![\mathbf{0}]\!]\rho s H = \emptyset$$

$$[\![P \mid Q]\!]\rho s H = [\![P]\!]\rho s H \cup [\![Q]\!]\rho s H$$

$$[\![!P]\!]\rho s H = [\![P]\!]\rho(s,i)H \text{ where } i \text{ is a fresh variable}$$

$$[\![\mathsf{new}\ a; P]\!]\rho s H = [\![P]\!](\rho[a \mapsto a[s]])s H$$

$$[\![\mathsf{in}(M,x); P]\!]\rho s H = [\![P]\!](\rho[x \mapsto x'])(s,x')(H \wedge \mathsf{message}(\rho(M),x'))$$

where x' is a fresh variable

$$[\![\mathsf{out}(M,N); P]\!]\rho s H = [\![P]\!]\rho s H \cup \{H \Rightarrow \mathsf{message}(\rho(M),\rho(N))\}$$

6. Horn子句

ρ represents the names and variables that are already associated with a pattern

- For the replication, we create a fresh session identifier ***i*** and add it to the sequence ***s***. The replication is otherwise ignored, because all Horn clauses are applicable arbitrarily many times.
- For the restriction, we replace the restricted name a in question with the pattern $a[s]$, where the sequence s contains the previous inputs and session identifiers.
- The sequence H is extended in the translation of an input, with the input in question. The sequence S is also extended with the received message.
- The translation of an output adds a clause, meaning that the output is triggered when all conditions in H are true.

6. Horn子句

Example:

$$P_0 = \mathsf{new}\ ssk_A : \mathsf{skey}; \mathsf{new}\ sk_B : \mathsf{skey}; \mathsf{let}\ spk_A = \mathsf{pk}(ssk_A)\ \mathsf{in}$$
$$\mathsf{let}\ pk_B = \mathsf{pk}(sk_B)\ \mathsf{in}\ \mathsf{out}(c, spk_A); \mathsf{out}(c, pk_B);$$
$$(P_A(ssk_A, pk_B) \mid P_B(sk_B, spk_A))$$

From P_0, we can obtain a Horn clause:

$$\mathsf{attacker}(\mathsf{pk}(ssk_A[\,])) \tag{3.1}$$
$$\mathsf{attacker}(\mathsf{pk}(sk_B[\,])) \tag{3.2}$$

6. Horn子句

Example:

$$P_A(ssk_A, pk_B) = !\,\mathsf{in}(c, x_{pk_B} : \mathsf{pkey}); \mathsf{new}\ k : \mathsf{key};$$
$$\mathsf{out}(c, \mathsf{aenc}(\mathsf{sign}(\mathsf{k2b}(k), ssk_A), x_{pk_B}));$$
$$\mathsf{in}(c, x : \mathsf{bitstring}); \mathsf{let}\ z = \mathsf{sdec}(x, k)\ \mathsf{in}\ \mathbf{0}$$

From P_A, we can obtain a Horn clause:

$$\mathsf{attacker}(x_{pk_B}) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(k[i, x_{pk_B}], ssk_A[\,]), x_{pk_B})) \tag{3.3}$$

6. Horn子句

Example:

$$P_B(sk_B, spk_A) = !\,\mathsf{in}(c, y : \mathsf{bitstring}); \mathsf{let}\ y' = \mathsf{adec}(y, sk_B)\ \mathsf{in}$$
$$\mathsf{let}\ x_k = \mathsf{b2k}(\mathsf{check}(y', spk_A))\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(s, x_k))$$

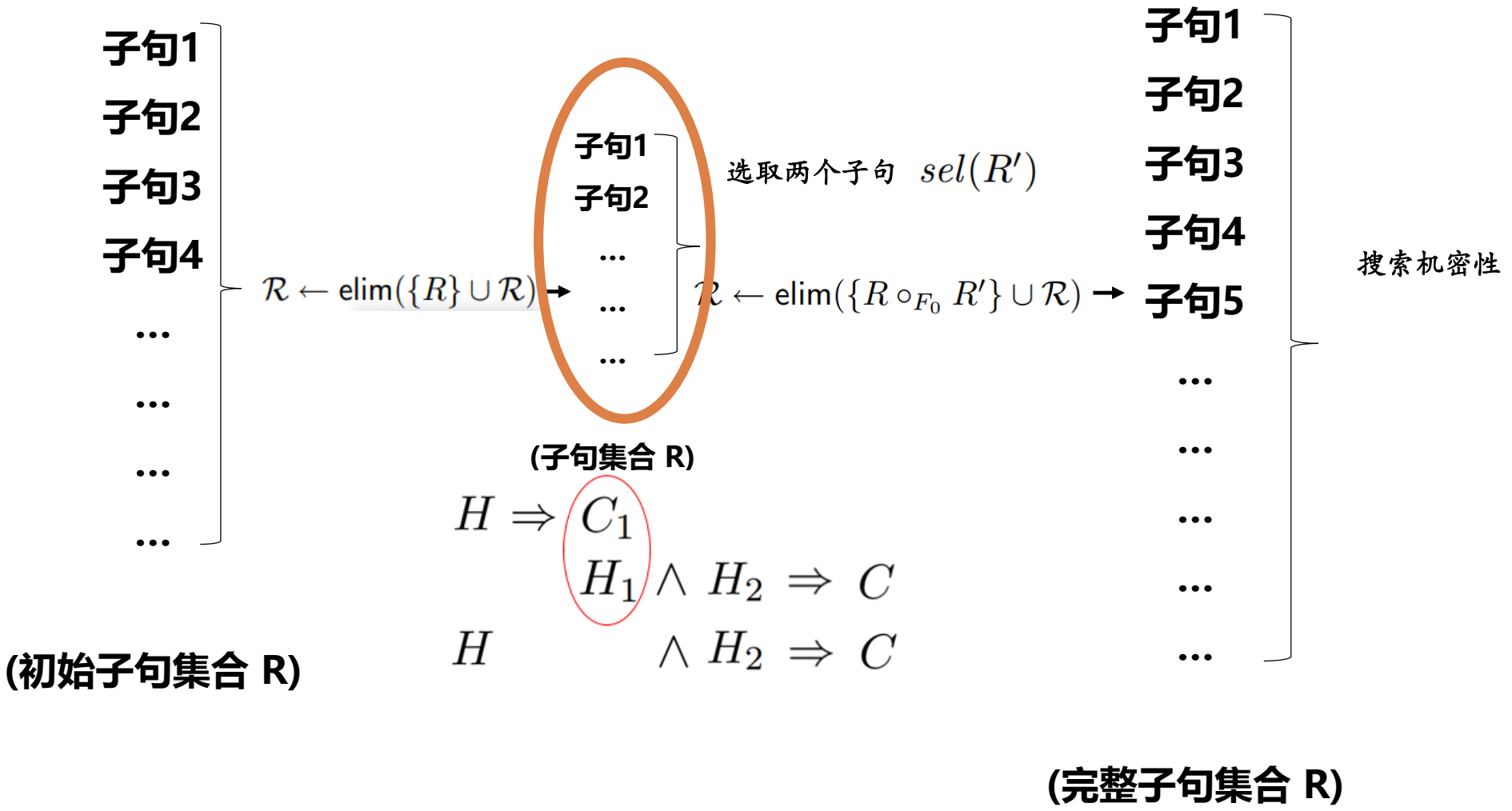
From P_B, we can obtain a Horn clause:

$$\mathsf{attacker}(\mathsf{aenc}(\mathsf{sign}(x_m, ssk_A[\,]), \mathsf{pk}(sk_B[\,]))) \Rightarrow \mathsf{attacker}(\mathsf{senc}(s[\,], x_m)) \tag{3.4}$$

Totally, $\mathbf{attacker}(s[])$ is derivable from the above clauses.

6. Horn子句

3）Resolution消解算法



6. Horn子句

3）Resolution 消解算法示例

1. attacker(aenc(m_45,pk(sk_46))) && attacker(sk_46) -> attacker(m_45)
2. attacker(pkX_100) -> attacker(aenc(s[],pkX_100))

$$H \Rightarrow C_1$$

$$H_1 \wedge H_2 \Rightarrow C$$

$$H \qquad \wedge H_2 \Rightarrow C$$

2. attacker(pkX_100) -> attacker(aenc(**s[]**, pkX_100))

↓ 1. attacker(aenc(**m_45**, pk(sk_46))) && attacker(sk_46) -> attacker(m_45)

↓ ↓ ↓

attacker(pk(sk_144)) && attacker(sk_144) -> attacker(s[])

m_45用s[]代替

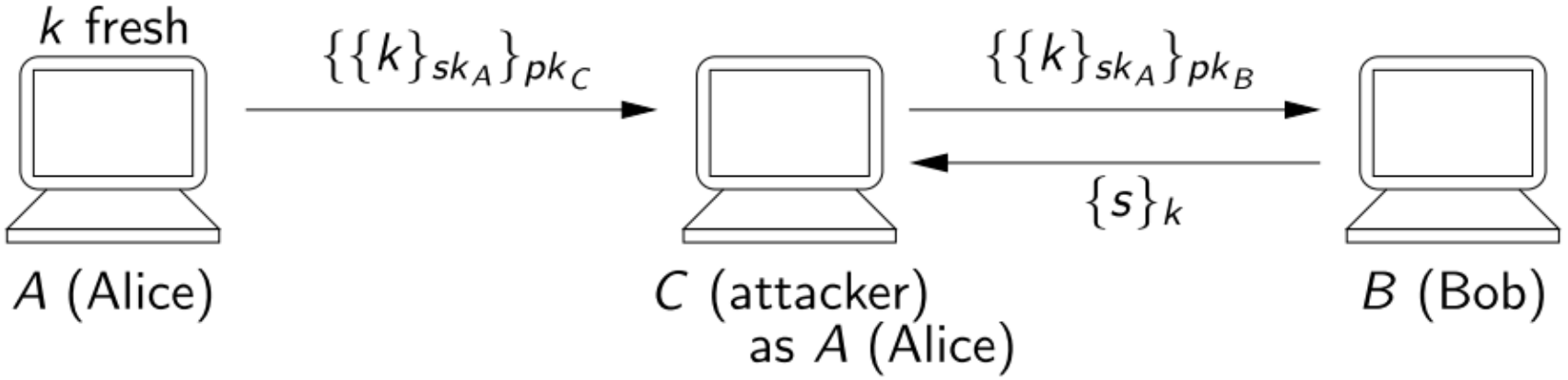
pkX_100和pk(sk_46)统一用pk(sk_144)代替。

6. Horn子句

Totally, attacker(s[]) is derivable from the above clauses.

(参见示例文件)

- The well-known attack (Abadi and Needham, 1996) against this protocol:



- The attacker C impersonates A and obtains the secret s.

7. FIDO UAF协议的形式化分析

- 发表在NDSS 2021
- 利用ProVerif对FIDO UAF进行全面分析

A Formal Analysis of the FIDO UAF Protocol

Haonan Feng[1], Hui Li[1], Xuesong Pan[1], Ziming Zhao[2]

[1]Beijing University of Posts and Telecommunications

[2]CactiLab, University at Buffalo

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

University at Buffalo

8. ProVerif总结

1)通过应用π-演算描述协议

2)描述协议安全属性

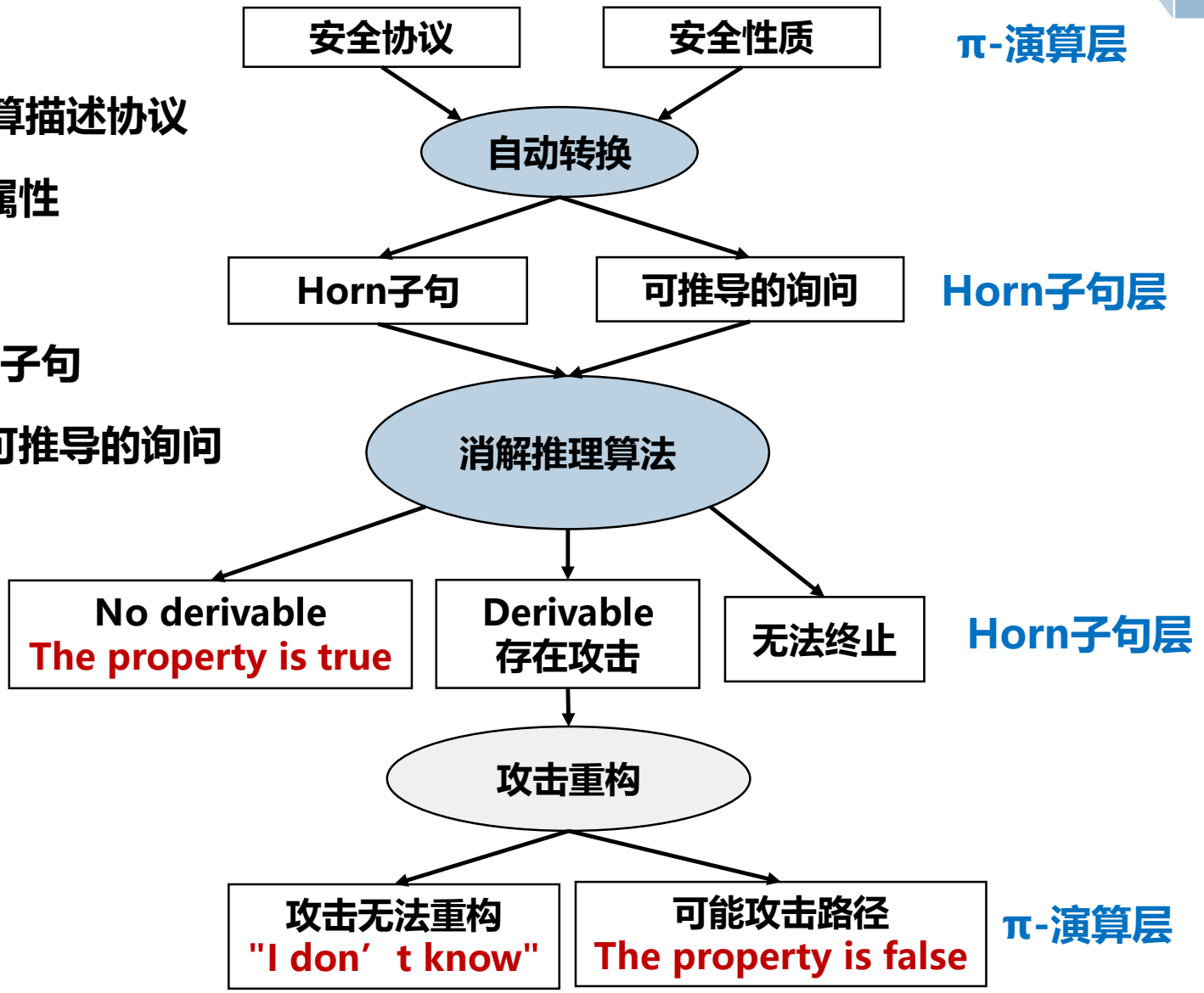
3)进行自动转换

4)协议转为Horn子句

5)安全性质转为可推导的询问

6)消解推理算法

7)攻击重构



8. ProVerif总结

- **Strong points of ProVerif:**
  - Automatic.
  - Proves protocols for an unbounded number of sessions and unbounded message space.
  - Handles a wide variety of primitives defined rewrite rules or equations.
  - Proves a wide variety of security properties (secrecy, authentication, equivalences).
  - Precise and efficient in practice.
- **Limitations:**
  - Risk of non-termination.
  - Risk of false attacks.
  - Still limited on the equations it supports and the equivalences it can prove.

8. ProVerif总结

Proverif作为后端构建其他协议验证工具

- TulaFale网络服务协议验证工具，处理XML消息
- JAVASPI Framework可以将Java语言转化为Proverif语言对协议进行安全验证
- Goubault-Larrecq等将C语言描述的协议转化为Proverif语言进行安全协议验证
- Web-spi library对web安全机制与协议进行建模并进行验证

谢谢大家！欢迎提问！