02244 Logic for Security
Security Protocols
The Dolev and Yao Intruder Model

Sebastian Mödersheim

February 5, 2024

Plan for Today

- Recap of last week and something new
- Completion of the Dolev-Yao intruder model
- A decision procedure for Dolev-Yao deductions
- Putting the intruder model to work
  - Outlook on the lazy intruder
- Hand out of the mandatory assignment.

Protocol from last week

```
B->A: NB
A->s: A,B,NA,NB
s->A: {|A,B,KAB,NA,NB|}sk(A,s),
      {|A,B,KAB,NA,NB|}sk(B,s)
A->B: {|A,B,KAB,NA,NB|}sk(B,s)
```

```
SUMMARY:
  NO_ATTACK_FOUND
```

- The best way to solve replay is to use challenge repsonse:
  - ★ Participants create a fresh random number like `NA` and `NB`.
  - ★ They are included in encrypted messages to prove that the encryption is not older than the fresh numbers.

Protocol from last week

```
B->A: NB
A->s: A,B,NA,NB
s->A: {|A,B,KAB,NA,NB|}sk(A,s),
      {|A,B,KAB,NA,NB|}sk(B,s)
A->B: {|A,B,KAB,NA,NB|}sk(B,s)
```

```
SUMMARY:
  NO_ATTACK_FOUND
```

- The best way to solve replay is to use challenge repsonse:
  - ★ Participants create a fresh random number like `NA` and `NB`.
  - ★ They are included in encrypted messages to prove that the encryption is not older than the fresh numbers.
  - ★ We are done. However there is a better way to do this using Diffie-Hellman!

Sixth Version

```
Protocol: KeyExchange
Types: Agent A,B,s;
       Number X,Y,g,Payload;
       Function sk;
Knowledge: A: A,B,s,sk(A,s),g;
           B: A,B,s,sk(B,s),g;
           s: A,B,s,sk(A,s),sk(B,s),g;
Actions:
A->B: exp(g,X)
B->s: {| A,B,exp(g,X),exp(g,Y) |}sk(B,s)
s->A: {| A,B,exp(g,X),exp(g,Y) |}sk(A,s)
A->B: {| Payload |}exp(exp(g,X),Y)
Goals:
exp(exp(g,X),Y) secret between A,B;
Payload secret between A,B;
A authenticates B on exp(exp(g,X),Y);
B authenticates A on exp(exp(g,X),Y),Payload;
```

Sixth Version

```
A->B: exp(g,X)
B->s: {| A,B,exp(g,X),exp(g,Y) |}sk(B,s)
s->A: {| A,B,exp(g,X),exp(g,Y) |}sk(A,s)
A->B: {| Payload |}exp(exp(g,X),Y)
```

Diffie-Hellman:

- every agent generates a random X and Y
- they exchange $\exp(g, X) \mod p$ and $\exp(g, Y) \mod p$
  - ★ p is a large fixed prime number – we omit in OFMC
  - ★ g is a fixed generator of the group $\mathbb{Z}_p^\star$
  - ★ Both p and g are public
  - ★ we omit writing $\mod p$ in OFMC
- It is computationally hard to obtain X from $\exp(g, X) \mod p$
- However A and B have now a shared key $\exp(\exp(g, X), Y) \mod p = \exp(\exp(g, Y), X) \mod p$

Diffie-Hellman and ECDH

	Classic	
Group	$\mathbb{Z}_p^\star = \{1, \ldots, p-1\}$	
Group Op.	$\times : \mathbb{Z}_p^\star \times \mathbb{Z}_p^\star \to \mathbb{Z}_p^\star$ (Mult. modulo p)	
Generator	$g \in \mathbb{Z}_p^\star$	
Secrets	$X, Y \in \{1, \ldots, p-1\}$	
Half keys	$g^X := \underbrace{g \times \ldots \times g}_{X \text{ times}}$ $g^Y := \ldots$	
Full key	$(g^X)^Y = (g^Y)^X$	

Diffie-Hellman and ECDH

	Classic	Elliptic Curve (ECDH)
Group	$\mathbb{Z}_p^\star = \{1, \ldots, p-1\}$	Finite field $\mathbb{F}$ of order n
Group Op.	${\color{blue}\times} : \mathbb{Z}_p^\star \times \mathbb{Z}_p^\star \to \mathbb{Z}_p^\star$ (Mult. modulo p)	${\color{red}+} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (not quite so intuitive...)
Generator	$g \in \mathbb{Z}_p^\star$	g on curve
Secrets	$X, Y \in \{1, \ldots, p-1\}$	$X, Y \in \{1, \ldots, n-1\}$
Half keys	$g^X := \underbrace{g {\color{blue}\times} \ldots {\color{blue}\times} g}_{X \text{ times}}$ $g^Y := \ldots$	$X \cdot g := \underbrace{g {\color{red}+} \ldots {\color{red}+} g}_{X \text{ times}}$ $Y \cdot g := \ldots$
Full key	$(g^X)^Y = (g^Y)^X$	$X \cdot Y \cdot g = Y \cdot X \cdot g$

Diffie-Hellman and ECDH

	Classic	Elliptic Curve (ECDH)
Group	$\mathbb{Z}_p^\star = \{1, \ldots, p-1\}$	Finite field $\mathbb{F}$ of order n
Group Op.	$\times : \mathbb{Z}_p^\star \times \mathbb{Z}_p^\star \to \mathbb{Z}_p^\star$ (Mult. modulo p)	$\times : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (not quite so intuitive...)
Generator	$g \in \mathbb{Z}_p^\star$	g on curve
Secrets	$X, Y \in \{1, \ldots, p-1\}$	$X, Y \in \{1, \ldots, n-1\}$
Half keys	$g^X := \underbrace{g \times \ldots \times g}_{X \text{ times}}$ $g^Y := \ldots$	$g^X := \underbrace{g \times \ldots \times g}_{X \text{ times}}$ $g^Y := \ldots$
Full key	$(g^X)^Y = (g^Y)^X$	$(g^X)^Y = (g^Y)^X$

Trick: write $\times$ for the group operation also in ECDH.

Diffie-Hellman and ECDH

	Classic	Elliptic Curve (ECDH)
Group	$\mathbb{Z}_p^\star = \{1, \ldots, p-1\}$	Finite field $\mathbb{F}$ of order n
Group Op.	$\times : \mathbb{Z}_p^\star \times \mathbb{Z}_p^\star \to \mathbb{Z}_p^\star$ (Mult. modulo p)	$\times : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (not quite so intuitive...)
Generator	$g \in \mathbb{Z}_p^\star$	g on curve
Secrets	$X, Y \in \{1, \ldots, p-1\}$	$X, Y \in \{1, \ldots, n-1\}$
Half keys	$g^X := \underbrace{g \times \ldots \times g}_{X \text{ times}}$ $g^Y := \ldots$	$g^X := \underbrace{g \times \ldots \times g}_{X \text{ times}}$ $g^Y := \ldots$
Full key	$(g^X)^Y = (g^Y)^X$	$(g^X)^Y = (g^Y)^X$
Typical size	thousand of bits	hundreds of bits

Trick: write $\times$ for the group operation also in ECDH.

Modeling Agents and Fixed Key-Infrastructures

- Normally variables (uppercase) like A,B,C,...
  - ★ can be played by any concrete (lowercase) agent like a,b,c,...,i
- Special agent: i – the intruder
- Honest agent: constant like s for a trusted server
  - ★ Cannot be instantiated (especially the intruder), fixed in all protocol runs
- Given key infrastructures: use functions e.g.
  - ★ sk(A,B) the shared key of A and B
  - ★ pw(A,B) the password of A at server B
  - ★ pk(A) the public key of A
    - ▶ inv(K) is the private key that belongs to public key K.
    - ▶ Note inv and exp are a built-in function (do not declare as a function).
  - ★ Give every role the necessary initial knowledge

AnB: Things to Note

- Identifiers that start with uppercase: variables (E.g., `A,B,KAB`)
- Identifiers that start with lowercase: constants and functions (E.g., `s,pre,sk`)
- One should declare a type for all identifiers; OFMC can search for *type-flaw* attacks when using the option `-untyped` (in which case all types are ignored).
- The (initial) knowledge of agents MUST NOT contain variables of any type other than `Agent`.
  - For long-term keys, passwords, etc. use functions like $sk(A, B)$.
- Each variable that does not occur in the initial knowledge is freshly created during the protocol by the first agent who uses it.
  - In the NSSK example, `A` creates `NA`, `s` creates `KAB`, `B` creates `NB`.

Message Term Algebra

for security protocols

Symbol	Arity	Meaning	Public
i	0	name of the intruder	yes
inv	1	private key of a given public key	no
crypt	2	asymmetric encryption in AnB: write $\{m\}_k$ for crypt(k, m)	yes
scrypt	2	symmetric encryption in AnB: write $\{\|m\|\}_k$ for scrypt(k, m)	yes
pair	2	pairing/concatenation in AnB: write m, n for pair(m, n)	yes
$\exp(\cdot, \cdot)$	2	exponentiation modulo fixed prime p	yes
$a, b, c, \ldots$	0	User-defined constants	User-def.
$f(\cdot)$	$\star$	User-defined function symbol f	User-def.

- Call Σ the set of all function symbols and Σ_p the public ones.
- Public functions can be applied by every agent
- inv is not public: the private key of a given public key.

Intruder Deduction

The core of the Dolev-Yao model is a definition what the intruder can do with messages.

- We define a relation $M \vdash m$ where
  - ★ M is a set of messages
  - ★ m is a message

  expressing that the intruder can derive m, if his knowledge is M.

Example

$$M = \{ k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \}$$

Then we should have for instance:

- $M \vdash m_1$
- $M \vdash m_2$
- $M \nvdash m_3$
- $M \vdash \{\!|\langle m_1, m_2\rangle|\!\}_{k_1}$

Dolev-Yao Closure

We define $M \vdash t$ as a proof calculus with rules of the form

$$\frac{\mathit{Premise}_1 \quad \ldots \quad \mathit{Premise}_n}{\mathit{Conclusion}} \ \mathit{Side\text{-}Condition}$$

meaning:

- if we have proved all the premisses
- and the side-condition holds,
- then we have a proof of the conclusion.

The simplest rule is Axiom:

Axiom

$$\frac{}{M \vdash m} \text{ if } m \in M \text{ (Axiom)}$$

The intruder can derive every message m that is directly in his knowledge M.

Free Exercise Today

Design your first own proof calculus!

Here are the first two rules to characterize Dolev-Yao's $M \vdash m$:

Axiom

The intruder can derive any message m that is already in his knowledge M:

$$\frac{}{M \vdash m}\ m \in M$$

Symmetric Decryption

The intruder can decrypt the message $\{|m|\}_k$ if he can derive k, and thus obtain the content m:

$$\frac{M \vdash \{|m|\}_k \quad M \vdash k}{M \vdash m}$$

Define similar rules for symmetric encryption, asymmetric encryption/decryption, signatures/signature verification, hashing, pair and obtaining the components of a pair.

Dolev-Yao Closure: Symmetric Crypto

Symmetric Cryptography

$$\frac{M \vdash m \quad M \vdash k}{M \vdash \{\!|m|\!\}_k} \text{ (EncSym)} \qquad \frac{M \vdash \{\!|m|\!\}_k \quad M \vdash k}{M \vdash m} \text{ (DecSym)}$$

- The intruder can encrypt any message m he knows with any key k he knows.
- The intruder can decrypt any message $\{\!|m|\!\}_k$ to which he knows the decryption key k.

Example

$$M = \{\, k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\}$$

$$\frac{\dfrac{}{M \vdash \{\!|m_1|\!\}_{k_1}}\text{ Axiom} \quad \dfrac{}{M \vdash k_1}\text{ Axiom}}{M \vdash m_1}\text{ DecSym}$$

Note: $M \nvdash m_3$

Dolev-Yao Closure: Concatenation

Concatenation

$$\frac{M \vdash m_1 \quad M \vdash m_2}{M \vdash \langle m_1, m_2 \rangle}\ (\mathrm{Cat}) \qquad \frac{M \vdash \langle m_1, m_2 \rangle}{M \vdash m_i}\ (\mathrm{Proj}_i)$$

- The intruder can concatenate and split messages.

Example

$$M = \{\, k_1, \{\!|\langle a, m_1 \rangle|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\}$$

$$\cfrac{\cfrac{\cfrac{\cfrac{}{M \vdash \{\!|\langle a, m_1 \rangle|\!\}_{k_1}}\ \mathrm{Axiom} \qquad \cfrac{}{M \vdash k_1}\ \mathrm{Axiom}}{M \vdash \langle a, m_1 \rangle}\ \mathrm{DecSym}}{M \vdash m_1}\ \mathrm{Proj}_2 \qquad \cfrac{}{M \vdash m_2}\ \mathrm{Axiom}}{M \vdash \langle m_1, m_2 \rangle}\ \mathrm{Cat}$$

Dolev-Yao Closure: Asymmetric Crypto

Asymmetric Cryptography

$$\frac{M \vdash m \quad M \vdash k}{M \vdash \{m\}_k} \text{ (EncAsym)} \qquad \frac{M \vdash \{m\}_k \quad M \vdash \mathsf{inv}(k)}{M \vdash m} \text{ (DecAsym)}$$

- The intruder can encrypt any message m he knows with any public key k he knows.
- The intruder can decrypt any message $\{m\}_k$ if he knows the private key $\mathsf{inv}(k)$ to the public key k.

Example

$$M = \{\, k_1, \mathsf{inv}(k_1), k_2, \{m_1\}_{k_1}, m_2, \{m_3\}_{k_2} \,\}$$

$$\dfrac{\dfrac{}{M \vdash \{m_1\}_{k_1}}\text{ Axiom} \qquad \dfrac{}{M \vdash \mathsf{inv}(k_1)}\text{ Axiom}}{M \vdash m_1}\text{ DecAsym}$$

Note: $M \nvdash m_3$

Dolev-Yao Closure: Signatures

Signatures

$$\frac{M \vdash m \quad M \vdash \mathsf{inv}(k)}{M \vdash \{m\}_{\mathsf{inv}(k)}}\ (\mathsf{Sign}) \qquad \frac{M \vdash \{m\}_{\mathsf{inv}(k)}}{M \vdash m}\ (\mathsf{OpenSig})$$

- The intruder can sign any message m he knows with any private key $\mathsf{inv}(k)$ he knows.
- The intruder can open any message $\{m\}_{\mathsf{inv}(k)}$ that was signed with a private key $\mathsf{inv}(k)$.

Example

$$M = \{\, k_1, \mathsf{inv}(k_1), k_2, \{m_1\}_{\mathsf{inv}(k_2)}, m_2 \,\}$$

$$\dfrac{\dfrac{\dfrac{}{M \vdash \{m_1\}_{\mathsf{inv}(k_2)}}\ \mathsf{Axiom}}{M \vdash m_1}\ \mathsf{OpenSig} \qquad \dfrac{}{M \vdash \mathsf{inv}(k_1)}\ \mathsf{Axiom}}{M \vdash \{m_1\}_{\mathsf{inv}(k_1)}}\ \mathsf{Sign}$$

Dolev-Yao Closure: Public Functions

Public Functions

$$\frac{M \vdash m_1 \quad \ldots \quad M \vdash m_n}{M \vdash f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ takes } n \text{ arguments} \quad \text{(Compose)}$$

- The intruder can apply any public function f to terms t_i that he knows.

Example

$$M = \{\, k_1, k_2, \{\!|m_1|\!\}_{kdf(k_1,k_2)} \,\} \text{ where } kdf \text{ is public}$$

$$\dfrac{\dfrac{}{M \vdash \{\!|m_1|\!\}_{kdf(k_1,k_2)}}\text{Axiom} \qquad \dfrac{\dfrac{}{M \vdash k_1}\text{Axiom} \quad \dfrac{}{M \vdash k_2}\text{Axiom}}{M \vdash kdf(k_1, k_2)}\text{Compose}}{M \vdash m_1}\text{DecSym}$$

Note: the rules EncSym, EncAsym, and Cat are just special cases of Compose.

Dolev-Yao Closure: Summary

Dolev-Yao rules

$$\frac{}{M \vdash m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash m_1 \quad \ldots \quad M \vdash m_n}{M \vdash f(m_1, \ldots, m_n)} \text{ if } f/n \in \Sigma_p \text{ (Compose)}$$

$$\frac{M \vdash \langle m_1, m_2 \rangle}{M \vdash m_i} \text{ (Proj}_i\text{)} \qquad \frac{M \vdash \{\!|m|\!\}_k \quad M \vdash k}{M \vdash m} \text{ (DecSym)}$$

$$\frac{M \vdash \{m\}_k \quad M \vdash \mathrm{inv}(k)}{M \vdash m} \text{ (DecAsym)} \qquad \frac{M \vdash \{m\}_{\mathrm{inv}(k)}}{M \vdash m} \text{ (OpenSig)}$$

The compose rule is for all public functions Σ_p,
including $\{\!| \cdot |\!\}.$ $\quad \{\cdot\}.$ $\quad \langle \cdot, \cdot \rangle$

Example: Intruder Deduction

Example

$M = \{\ a, b, i, \mathsf{pk}(a), \mathsf{pk}(b), \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), \{\langle na, a\rangle\}_{\mathsf{pk}(i)}\}$

Can the intruder derive $\{\langle na, a\rangle\}_{\mathsf{pk}(b)}$?

Automation

Goal: design (in pseudocode) a decision procedure for Dolev-Yao:

- Given a finite set M of messages (the intruder knowledge)
- and given a message m (the goal)
- Output whether $M \vdash m$ holds.
  - ★ additionally, in the positive case, give the proof.

How to approach this?

Automating Dolev-Yao

Step 1: Composition only

Consider first the following simpler problem:

- $M \vdash_c m$ are those deductions where the intruder does not apply any analysis steps ("composition only"):

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{|m|\}_{h(k_1,k_2)}\}$ where $h \in \Sigma_p$

- $M \vdash_c h(k_1, k_2)$
- $M \nvdash_c m$
- $M \vdash m$

Automating Dolev-Yao

To check $M \vdash m$:

- Perform the Analysis Steps procedure, augmenting M with all derivable messages.
- Now it suffices to check $M \vdash_c m$.

Properties of the algorithm for checking $M \vdash m$:

- Soundness: if algorithm says "yes", then $M \vdash m$.
- Completeness: if $M \vdash m$, then the algorithm says "yes".
  - ★ This is quite tricky to prove.
- Termination: the algorithm never runs into an infinite loop.

Negative Question

Can we thus prove also statements of the form $M \nvdash m$

...that a m cannot be derived from M?

Example

$$M = \{\, k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\} \nvdash m_3$$

Negative Question

Can we thus prove also statements of the form $M \nvdash m$
... that a m cannot be derived from M?

Example

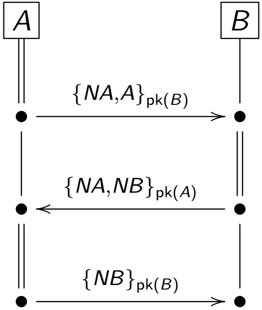
$$M = \{\, k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\} \nvdash m_3$$

- Yes, due to completeness when our algorithm answers "no", we know there is no derivation for m.

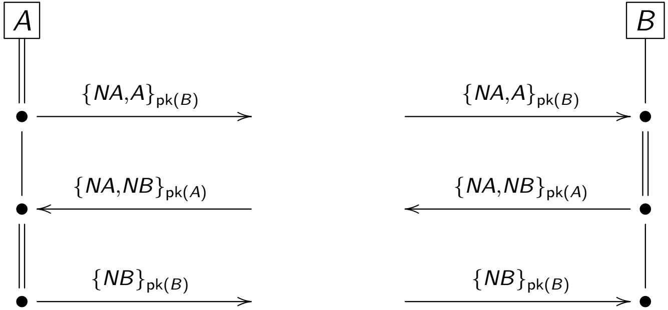
Needham-Schroeder Public-Key Protocol [1978]

```
Protocol: NSPK

Types: Agent A,B;
       Number NA,NB;
       Function pk,h

Knowledge: A: A,pk(A),inv(pk(A)),B,pk(B),h;
           B: B,pk(B),inv(pk(B)),A,pk(A),h
Actions:
A->B: {NA,A}(pk(B))  # A generates NA
B->A: {NA,NB}(pk(A)) # B generates NB
A->B: {NB}(pk(B))

Goals:
h(NA,NB) secret between A,B
```

NSPK as A Message Sequence Chart

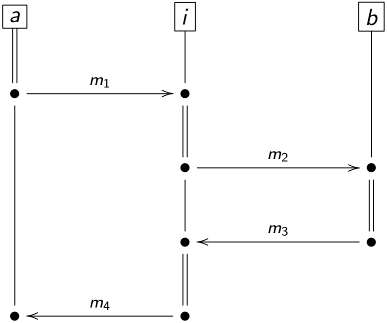


NSPK as Roles / Strands



- For each Role of the protocol, a program that sends and receives messages (over possibly insecure network)
- Strand: concrete execution of a role: all variables (here A, B, NA, NB) instantiated with concrete values
  - ★ or a prefix thereof (an agent might not finish)

Attacks

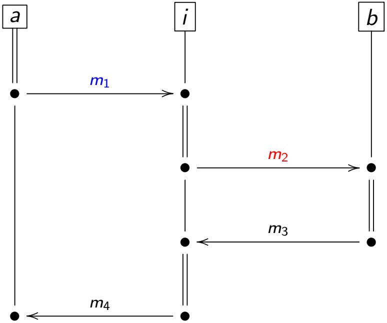


An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by i
- Messages received by honest agents are sent by i who can compose the message from the messages he has received so far.

- The successful completion violates a goal of the protocol.

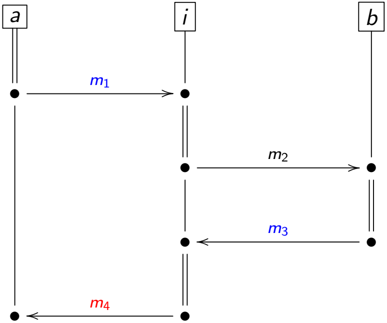
Attacks



An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by i
- Messages received by honest agents are sent by i who can compose the message from the messages he has received so far.
  - ★ In the example: $\{m_1\} \vdash m_2$
- The successful completion violates a goal of the protocol.

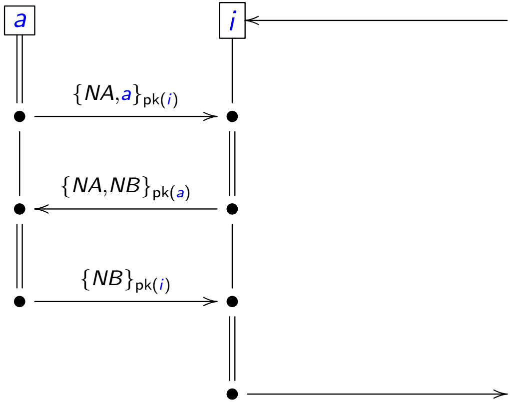
Attacks



An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by i
- Messages received by honest agents are sent by i who can compose the message from the messages he has received so far.
  - ★ In the example: $\{m_1\} \vdash m_2$ and $\{m_1, m_3\} \vdash m_4$.
- The successful completion violates a goal of the protocol.

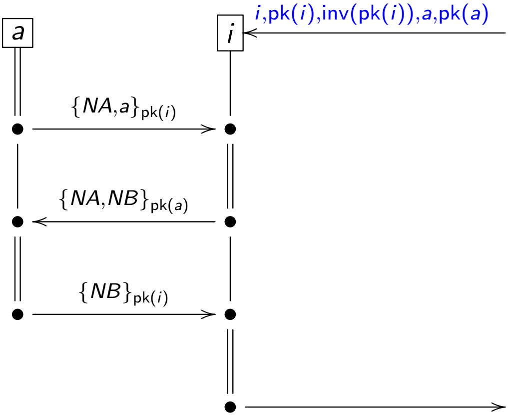
Example: Can the Intruder Play a Protocol Role?



Example:

- NSPK with $A = a$ and $B = i$

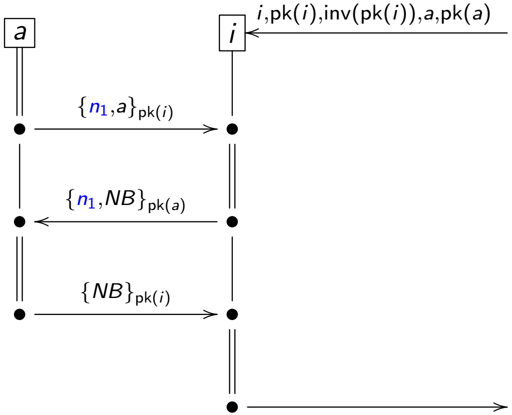
Example: Can the Intruder Play a Protocol Role?



Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$

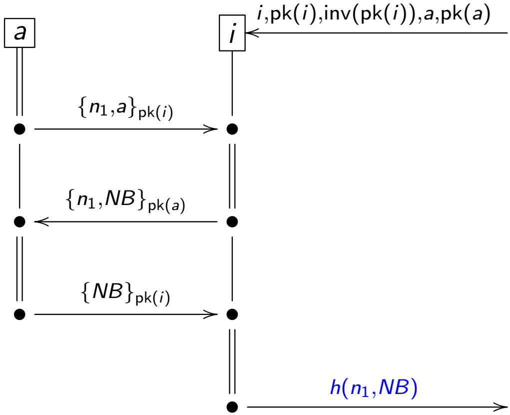
Example: Can the Intruder Play a Protocol Role?



Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathrm{pk}(i), \mathrm{inv}(\mathrm{pk}(i)), a, \mathrm{pk}(a)$
- a uses a fresh $NA = n_1$.

Example: Can the Intruder Play a Protocol Role?



Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$
- a uses a fresh $NA = n_1$.
- Afterwards, i should be able to construct the shared key $h(NA, NB) = h(n_1, NB)$

Challenge

Consider the following protocol:

```
A -> B : A,B,exp(g,X)
B -> A: { A,B,exp(g,X),exp(g,Y) }inv(pk(B)),
        { B, pk(B) }inv(pk(s))
A->B: A,B,{| Msg, pw(A,B) |} exp(  exp(g,Y) , X)
```

What should the strands for A and B look like?
Suppose $A = i$ and $B = b$, show that the intruder can do all the steps for the A role (similar to how we showed it for NSPK for the B role) if the intruder initially knows $i, b, pw(i, b), g, pk(s)$.

Relevant Research Papers

- David Basin, Sebastian Mödersheim, and Luca Viganò. *OFMC: A symbolic model checker for security protocols.* International Journal of Information Security, 4(3), 2005.
- Gavin Lowe. *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*. Software Concepts Tools, 17(3), 1996.
- Jonathan K. Millen and Vitaly Shmatikov. *Constraint solving for bounded-process cryptographic protocol analysis*. Computer and Communications Security, 2001,
- Roger Needham and Michael Schroeder. *Using Encryption for Authentication in Large Networks of Computers.* Communications of the ACM, 21(12), 1978.
- Michaël Rusinowitch and Mathieu Turuani. *Protocol Insecurity with Finite Number of Sessions is NP-complete.* Computer Security Foundations Workshop, 2001.