Certification of Cryptographic Protocols by Model-checking and Proof Concretization

R. Janvier, Y. Lakhnech, M. Périn

VERIMAG Lab.
Grenoble, France

4 April 2006
Workshop on Innovative Techniques for Certification of Embedded Systems
The open system \((e.g.,\ a\ smart\ card)\) ensures security based on the assumptions provided by the cryptographic protocols used in exchange with the environment \((e.g.,\ the\ smart\ card\ interface)\).
Cryptographic protocols are used to ensure authentication, non-repudiation, anonymity,… almost all rely on the secrecy property in a hostile environment a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones Dolev and Yao’s inference rules define the deduction capacities of the intruder, e.g, he can only decipher an encrypted message if he knows the decryption key We assume perfect cryptography and we are interested in logical flaws of protocols. We do not consider attack on cryptographic algorithms (the mathematical level).
Cryptographic protocols are used to ensure authentication, non-repudiation, anonymity,... almost all rely on the secrecy property in a hostile environment where a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones. Dolev and Yao’s inference rules define the deduction capacities of the intruder, e.g., he can only decipher an encrypted message if he knows the decryption key.

We assume perfect cryptography and we are interested in logical flaws of protocols.

We do not consider attack on cryptographic algorithms (the mathematical level).
Cryptographic protocols

are used to ensure

authentication, non-repudiation, anonymity,...

almost all rely on the secrecy property

in a hostile environment

a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones

*Dolev and Yao’s inference rules define the deduction capacities of the intruder*, e.g, he can only decipher an encrypted message if he knows the decryption key

We assume perfect cryptography and we are interested in logical flaws of protocols.

We do not consider attack on cryptographic algorithms (the mathematical level).
Needham-Schroeder’s authentication protocol

The goal of this protocol

Mutual authentication of principals $A$ and $B$ based on exchange of a shared secret: the randomly generated number $N_B$

<table>
<thead>
<tr>
<th>Needham-Schroeder</th>
<th>Needham-Schroeder-Lowe</th>
<th>Bad correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $A \rightarrow B$:</td>
<td>$!{A, N_A}_{k_B}$</td>
<td></td>
</tr>
<tr>
<td>2 $B \rightarrow A$:</td>
<td>$?{x, n_a}<em>{k_B}$ $\rightarrow$ $!{n_a, N_B}</em>{k_x}$</td>
<td>attack</td>
</tr>
<tr>
<td>2 $B \rightarrow A$:</td>
<td>$?{x, n_a}<em>{k_B}$ $\rightarrow$ $!{B, (n_a, N_B)}</em>{k_x}$</td>
<td>safe</td>
</tr>
<tr>
<td>2 $B \rightarrow A$:</td>
<td>$?{x, n_a}<em>{k_B}$ $\rightarrow$ $!{n_a, (N_B, B)}</em>{k_x}$</td>
<td>attack</td>
</tr>
<tr>
<td>3 $A \rightarrow B$:</td>
<td>$?{N_A, n_b}<em>{k_A}$ $\rightarrow$ $!{n_b}</em>{k_B}$</td>
<td>Man in the middle</td>
</tr>
<tr>
<td>3 $A \rightarrow B$:</td>
<td>$?{B, (N_A, n_b)}<em>{k_A}$ $\rightarrow$ $!{n_b}</em>{k_B}$</td>
<td></td>
</tr>
<tr>
<td>3 $A \rightarrow B$:</td>
<td>$?{N_A, \underbrace{(n_b, B)}<em>{n_b}}</em>{k_A}$ $\rightarrow$ $!{n_b}_{k_B}$</td>
<td>Type attack + m.i.m</td>
</tr>
</tbody>
</table>
Needham-Schroeder’s authentication protocol

The goal of this protocol
Mutual authentication of principals $A$ and $B$ based on exchange of a shared secret: the randomly generated number $N_B$

<table>
<thead>
<tr>
<th></th>
<th>Needham-Schroeder</th>
<th>Needham-Schroeder-Lowe</th>
<th>Bad correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A \to B$:</td>
<td>${ A, N_A }_{k_B}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$B \to A$:</td>
<td>${ x, n_a }_{k_B}$</td>
<td>${ n_a, N_B }_{k_x}$ attack</td>
</tr>
<tr>
<td>2</td>
<td>$B \to A$:</td>
<td>${ x, n_a }_{k_B}$</td>
<td>${ B, (n_a, N_B) }_{k_x}$ safe</td>
</tr>
<tr>
<td>2</td>
<td>$B \to A$:</td>
<td>${ x, n_a }_{k_B}$</td>
<td>${ n_a, (N_B, B) }_{k_x}$ attack</td>
</tr>
<tr>
<td>3</td>
<td>$A \to B$:</td>
<td>${ N_A, n_b }_{k_A}$</td>
<td>${ n_b }_{k_B}$ Man in the middle</td>
</tr>
<tr>
<td>3</td>
<td>$A \to B$:</td>
<td>${ B, (N_A, n_b) }_{k_A}$</td>
<td>${ n_b }_{k_B}$ Type attack + m.i.m</td>
</tr>
<tr>
<td>3</td>
<td>$A \to B$:</td>
<td>${ N_A, (n_b, B) }_{k_A}$</td>
<td>${ n_b }_{k_B}$</td>
</tr>
</tbody>
</table>

R. Janvier, Y. Lakhnech, M. Pépin
Certification of Cryptographic Protocols
### Needham-Schroeder’s authentication protocol

**The goal of this protocol**

Mutual authentication of principals $A$ and $B$ based on exchange of a shared secret: the randomly generated number $N_B$

### Needham-Schroeder vs. Needham-Schroeder-Lowe

<table>
<thead>
<tr>
<th>Step</th>
<th>Needham-Schroeder</th>
<th>Needham-Schroeder-Lowe</th>
<th>Bad correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A \rightarrow B$: &amp; $\rightarrow {A, N_A}_{kB}$ &amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$B \rightarrow A$: &amp; $\rightarrow {n_a, N_B}_{k_x}$ &amp; attack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$B \rightarrow A$: &amp; $\rightarrow {B, (n_a, N_B)}_{k_x}$ &amp; safe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow B$: &amp; $\rightarrow {n_a, (N_B, B)}_{k_x}$ &amp; attack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow B$: &amp; $\rightarrow {n_b}_{kB}$ &amp; Man in the middle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow B$: &amp; $\rightarrow {n_b}_{kB}$ &amp; Type attack + m.i.m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow B$: &amp; $\rightarrow {n_b}_{kB}$ &amp;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R. Janvier, Y. Lakhnech, M. Périn

Certification of Cryptographic Protocols
The goal of this protocol

Mutual authentication of principals $A$ and $B$ based on exchange of a shared secret: the randomly generated number $N_B$.

<table>
<thead>
<tr>
<th>Needham-Schroeder</th>
<th>Needham-Schroeder-Lowe</th>
<th>Bad correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B$: !${A, N_A}_{k_B}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B \rightarrow A$: ${x, n_a}<em>{k_B} \rightarrow !{n_a, N_B}</em>{k_x}$ attack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B \rightarrow A$: ${x, n_a}<em>{k_B} \rightarrow !{B, (n_a, N_B)}</em>{k_x}$ safe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B \rightarrow A$: ${x, n_a}<em>{k_B} \rightarrow !{n_a, (N_B, B)}</em>{k_x}$ attack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow B$: ${N_A, n_b}<em>{k_A} \rightarrow !{n_b}</em>{k_B}$ Man in the middle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow B$: ${B, (N_A, n_b)}<em>{k_A} \rightarrow !{n_b}</em>{k_B}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow B$: ${N_A, (n_b, B)}<em>{k_A} \rightarrow !{n_b}</em>{k_B}$ Type attack + m.i.m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R. Janvier, Y. Lakhnech, M. Pépin

Certification of Cryptographic Protocols
The man in the middle attack on NS’s protocol

\[ k_A^{-1}, \text{all } k_x \]
\[ k_l^{-1}, \text{all } k_x \]
\[ k_B^{-1}, \text{all } k_x \]

\[ N_A \rightarrow \{A, N_A\}_{k_l} \rightarrow I \rightarrow \{A, N_A\}_{k_B} \rightarrow N_A \rightarrow \{N_A, N_B\}_{k_A} \rightarrow N_B = \text{sec}(A, B) \]

\[ \{N_B\}_{k_l} \rightarrow \text{new } N_B \rightarrow N_B \]
Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- Very sensitive to minor changes
- Error prone and difficult to prove due to the underlying semantics:
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel against Dolev and Yao’s intruder
  - The Needham-Schroeder’s authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.
Needs for **Verification** of Cryptography Protocols

- Basic components for security functionalities
- Very sensitive to minor changes
- Error prone and difficult to prove due to the underlying semantics:
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel against Dolev and Yao's intruder
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.
Needs for Verification of Cryptography Protocols

- Basic components for security functionalities
- Very sensitive to minor changes
- Error prone and difficult to prove due to the underlying semantics:
  - A proof requires to consider an unbounded number of sessions of protocol running in parallel against Dolev and Yao’s intruder
  - The Needham-Schroeder’s authentication protocol has been used during 17 years before G. Lowe discovered the “man-in-the-middle” attack by model-checking two sessions of the protocol.
Needs for **Verification** of Cryptography Protocols

- Basic components for security functionalities
- Very sensitive to minor changes
- Error prone and difficult to prove due to the underlying semantics:
  - A proof requires to consider an *unbounded number of sessions of protocol* running in parallel *against Dolev and Yao’s intruder*
  - The Needham-Schroeder’s authentication protocol has been used during 17 years before G.Lowe discovered the “man-in-the-middle” attack by model-checking two sessions of the protocol.
Needs for **Verification** of Cryptography Protocols

- Basic components for security functionalities
- Very sensitive to minor changes
- Error prone and difficult to prove due to the underlying semantics:
  - A proof requires to consider an *unbounded number of sessions of protocol* running in parallel against Dolev and Yao’s intruder
  - The Needham-Schroeder’s authentication protocol has been used during 17 years before G.Lowe discovered the “man-in-the-middle” attack by model-checking two sessions of the protocol.
Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

**Don’t trust the verification tool**
- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

**Check its verdict**
- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the Coq proof-engine and proof-checker
Certification aims at reducing the Trusted Computing Base

Don’t trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the Coq proof-engine and proof-checker
Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

\textbf{Don’t trust the verification tool}
- Verification by (possibly erroneous) model-checker
- Using (possibly unsafe) abstractions

\textbf{Check its verdict}
- Instrument the verification tool
- Produce a verdict that can be checked independently of the verification tool
- Verdict = a proof in a formal setting where proof checking is just type checking the proof term
- Need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker
Needs for Certification of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don’t trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

R. Janvier, Y. Lakhnech, M. Périn

Certification of Cryptographic Protocols
Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

**Don’t trust the verification tool**

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

**Check its verdict**

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker
Needs for **Certification** of Cryptography Protocols

**Certification aims at reducing the Trusted Computing Base**

**Don’t trust the verification tool**
- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

**Check its verdict**
- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
  - verdict = a proof in a formal setting where proof checking is just type checking the proof term
  - need only to trust the proof-checker
- HERMES uses the **Coq** proof-engine and proof-checker
Certification aims at reducing the Trusted Computing Base

Don’t trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker
Certification aims at reducing the Trusted Computing Base

**Don’t trust the verification tool**
- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

**Check its verdict**
- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker

HERMES uses the COQ proof-engine and proof-checker
Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

---

**Don’t trust the verification tool**

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

---

**Check its verdict**

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the **COQ** proof-engine and proof-checker
The principle of the HERMES verification tool

Transition 3 of a session between $A$ and $I$ of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} S$$

Although the key $k_A$ is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret $S$ on transitions of $R = P \cup DY$.

HERMES’ operator $\text{pre}(R, \text{secrets})$ returns new secrets that reveal the given $\text{secrets}$.
The principle of the HERMES verification tool

Transition 3 of a session between $A$ and $I$ of NS protocol

$A \rightarrow I : \{N_A, n_b\}_{k_A} \rightarrow \{n_b\}_{k_I} S$

$A \rightarrow I : \{N_A, S\}_{k_A} \rightarrow \{S\}_{k_I} \vdash_{DY} S$

$I$ knows $k_I^{-1}$

Although the key $k_A$ is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret $S$ on transitions of $R = \mathcal{P} \cup DY$.

HERMES’ operator $pre(R, secrets)$ returns new secrets that reveal the given $secrets$. 
The principle of the HERMES verification tool

Transition 3 of a session between $A$ and $I$ of NS protocol

$A \rightarrow I : \ ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \quad S$

$A \rightarrow I : \ ?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I} \quad \vdash_{DY} \quad S$

instance of transition 3 with $n_b=S$

Although the key $k_A$ is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret $S$ on transitions of $R = \mathcal{P} \cup DY$.

HERMES’ operator $\text{pre}(R, secrets)$ returns new secrets that reveal the given secrets.
The principle of the HERMES verification tool

Transition 3 of a session between $A$ and $I$ of NS protocol

$A \rightarrow I : \ ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \quad S$

$A \rightarrow I : \ ?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I} \quad \vdash_{DY} \quad S$

instance of transition 3 with $n_b=S$

Although the key $k_A$ is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret $S$ on transitions of $R = \mathcal{P} \cup DY$.

HERMES’ operator $pre(R, secrets)$ returns new secrets that reveal the given secrets.
The principle of the HERMES verification tool

Transition 3 of a session between $A$ and $I$ of NS protocol

\[
A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \quad S
\]

\[
A \rightarrow I : \quad ?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I} \quad \vdash_{DY} S
\]

instance of transition 3 with $n_b=S$

Although the key $k_A$ is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret $S$ on transitions of $R = P \cup DY$.

HERMES’ operator $\text{pre}(R, \text{secrets})$ returns new secrets that reveal the given secrets.
Reachability analysis based on the operator \textit{pre}

1. Backward computation of secrets starting with $S$ until reaching a set $S_{\text{HERMES}}$ of secrets stable for \textit{pre}.

2. $S_{\text{HERMES}} \cap \text{Initial knowledge of } I = \emptyset$

Abstractions used in HERMES

- abstraction on principals: $A, B$ are honest, the others aren't $Others \simeq$ the intruder $I$
- That induces the abstraction on keys $k_A, k_B, k_I$
- unbounded number of sessions abstracted in sessions $\mathcal{P}(A, B) \parallel \mathcal{P}(A, I) \parallel \mathcal{P}(I, A) \parallel \mathcal{P}(I, B) \parallel \mathcal{P}(B, I) \parallel \mathcal{P}(I, I)$
- abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
- abstract representation of the infinite set of dangerous messages as patterns of messages
The HERMES verification tool

Reachability analysis based on the operator $pre$

1. Backward computation of secrets starting with $S$ until reaching a set $S_{HERMES}$ of secrets stable for $pre$.

2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

- abstraction on principals: $A, B$ are honest, the others aren't
  $Others \simeq$ the intruder $I$
  That induces the abstraction on keys $k_A, k_B, k_I$

- unbounded number of sessions abstracted in sessions
  $P(A, B) \parallel P(A, I) \parallel P(I, A) \parallel P(I, B) \parallel P(B, I) \parallel P(I, I)$

- abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$

- abstract representation of the infinite set of dangerous messages as patterns of messages
The HERMES verification tool

**Reachability analysis based on the operator** \( \text{pre} \)

1. Backward computation of secrets starting with \( S \) until reaching a set \( S_{\text{HERMES}} \) of secrets stable for \( \text{pre} \).
2. \( S_{\text{HERMES}} \cap \text{Initial knowledge of} \ I = \emptyset \)

**Abstractions used in HERMES**

- abstraction on principals: \( A, B \) are honest, the others aren't \( \text{Others} \simeq \) the intruder \( I \)
  That induces the abstraction on keys \( k_A, k_B, k_I \)
- unbounded number of sessions abstracted in sessions \( P(A, B) \parallel P(A, I) \parallel P(I, A) \parallel P(I, B) \parallel P(B, I) \parallel P(I, I) \)
- abstraction on random numbers \( N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II} \)
- abstract representation of the infinite set of dangerous messages as patterns of messages
Reachability analysis based on the operator \( \text{pre} \)

1. Backward computation of secrets starting with \( S \) until reaching a set \( S_{\text{HERMES}} \) of secrets stable for \( \text{pre} \).

2. \( S_{\text{HERMES}} \cap \) Initial knowledge of \( I = \emptyset \)

Abstractions used in HERMES

1. abstraction on principals: \( A, B \) are honest, the others aren’t \( \text{Others} \equiv \) the intruder \( I \)

   That induces the abstraction on keys \( k_A, k_B, k_I \)

2. unbounded number of sessions abstracted in sessions \( P(A, B) \parallel P(A, I) \parallel P(I, A) \parallel P(I, B) \parallel P(B, I) \parallel P(I, I) \)

3. abstraction on random numbers \( N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II} \)

4. abstract representation of the infinite set of dangerous messages as patterns of messages
### Reachability analysis based on the operator $\text{pre}$

1. Backward computation of secrets starting with $S$ until reaching a set $S_{\text{HERMES}}$ of secrets stable for $\text{pre}$.

2. $S_{\text{HERMES}} \cap$ Initial knowledge of $I = \emptyset$

### Abstractions used in HERMES

1. abstraction on principals: $A, B$ are honest, the others aren’t $\text{Others} \simeq$ the intruder $I$
   
   That induces the abstraction on keys $k_A, k_B, k_I$

2. unbounded number of sessions abstracted in sessions $\mathcal{P}(A, B) \parallel \mathcal{P}(A, I) \parallel \mathcal{P}(I, A) \parallel \mathcal{P}(I, B) \parallel \mathcal{P}(B, I) \parallel \mathcal{P}(I, I)$

3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$

4. abstract representation of the infinite set of dangerous messages as patterns of messages
Reachability analysis based on the operator $pre$

1. Backward computation of secrets starting with $S$ until reaching a set $S_{HERMES}$ of secrets stable for $pre$.
2. $S_{HERMES} \cap \text{Initial knowledge of } I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: $A, B$ are honest, the others aren’t $Others \simeq$ the intruder $I$
   That induces the abstraction on keys $k_A, k_B, k_I$
2. unbounded number of sessions abstracted in sessions
   $$\mathcal{P}(A, B) \parallel \mathcal{P}(A, I) \parallel \mathcal{P}(I, A) \parallel \mathcal{P}(I, B) \parallel \mathcal{P}(B, I) \parallel \mathcal{P}(I, I)$$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages
Reachability analysis based on the operator $pre$

1. Backward computation of secrets starting with $S$ until reaching a set $S_{HERMES}$ of secrets stable for $pre$.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: $A, B$ are honest, the others aren’t $Others \simeq$ the intruder $I$
   That induces the abstraction on keys $k_A, k_B, k_I$
2. unbounded number of sessions abstracted in sessions
   $P(A, B) \parallel P(A, I) \parallel P(I, A) \parallel P(I, B) \parallel P(B, I) \parallel P(I, I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages
Reachability analysis based on the operator $\text{pre}$

1. Backward computation of secrets starting with $S$ until reaching a set $S_{\text{HERMES}}$ of secrets stable for $\text{pre}$.
2. $S_{\text{HERMES}} \cap \text{Initial knowledge of } I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: $A, B$ are honest, the others aren’t $\text{Others} \simeq \text{the intruder } I$
   That induces the abstraction on keys $k_A, k_B, k_I$
2. unbounded number of sessions abstracted in sessions
   $\mathcal{P}(A, B) \parallel \mathcal{P}(A, I) \parallel \mathcal{P}(I, A) \parallel \mathcal{P}(I, B) \parallel \mathcal{P}(B, I) \parallel \mathcal{P}(I, I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages
Certifying the *abstract verdict* of HERMES

Does the abstract system \((R^\alpha, I_0^\alpha)\) satisfies a property \(\text{Secret}(S^\alpha)\)?

**Negative verdict**
HERMES produces a counter-example. It is a proof of 
\((R^\alpha, I_0^\alpha) \not\models \text{Secret}(S^\alpha)\).

**Positive verdict**
HERMES generates a checkable proof based on the principle of induction for *pre*-reachability analysis.

<table>
<thead>
<tr>
<th>Usage conditions of the protocol</th>
<th>Correctness of HERMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S^\alpha \subseteq S^\alpha_{HERMES})</td>
<td>(\text{pre}(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES})</td>
</tr>
<tr>
<td>(S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset)</td>
<td>(R^\alpha \ast (I_0^\alpha) \cap S^\alpha = \emptyset)</td>
</tr>
</tbody>
</table>

\(\hat{}\)

R. Janvier, Y. Lakhnech, M. Pépin

Certification of Cryptographic Protocols
Certifying the *abstract verdict* of HERMES

Does the abstract system \((R^\alpha, I_0^\alpha)\) satisfies a property \(\text{Secret}(S^\alpha)\)?

**Negative verdict**

HERMES produces a counter-example. It is a proof of
\[(R^\alpha, I_0^\alpha) \not\models \text{Secret}(S^\alpha).\]

**Positive verdict**

HERMES generates a *checkable proof* based on the *principle of induction for pre-reachability analysis*

---

<table>
<thead>
<tr>
<th><strong>trivial proof</strong></th>
<th>Usage conditions of the protocol</th>
<th>Correctness of HERMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S^\alpha \subseteq S^\alpha_{\text{HERMES}})</td>
<td>(S^\alpha_{\text{HERMES}} \cap I_0^\alpha = \emptyset)</td>
<td>(\text{pre}(R^\alpha, S^\alpha_{\text{HERMES}}) \subseteq S^\alpha_{\text{HERMES}})</td>
</tr>
<tr>
<td>(\text{pre}^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset)</td>
<td></td>
<td>(R^\alpha^*(I_0^\alpha) \cap S^\alpha = \emptyset)</td>
</tr>
</tbody>
</table>
Certifying the *abstract verdict* of HERMES

Does the abstract system \((R^\alpha, I_0^\alpha)\) satisfies a property \(\text{Secret}(S^\alpha)\)?

<table>
<thead>
<tr>
<th>Negative verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>HERMES produces a counter-example. It is a proof of ((R^\alpha, I_0^\alpha) \not\models \text{Secret}(S^\alpha)).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Positive verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>HERMES generates a <strong>checkable proof</strong> based on the <strong>principle of induction</strong> for <strong>pre-reachability analysis</strong></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
S^\alpha &\subseteq S^\alpha_{\text{HERMES}} \\
S^\alpha_{\text{HERMES}} \cap I_0^\alpha & = \emptyset \\
\text{pre}^*(R^\alpha, S^\alpha_{\text{HERMES}}) &\subseteq S^\alpha_{\text{HERMES}} \\
\text{pre}^*(R^\alpha, S^\alpha) \cap I_0^\alpha & = \emptyset & \equiv & R^\alpha^*(I_0^\alpha) \cap S^\alpha & = \emptyset
\end{align*}
\]
Certifying the abstract verdict of HERMES

Does the abstract system \((R^\alpha, I_0^\alpha)\) satisfies a property \(\text{Secret}(S^\alpha)\)?

### Negative verdict
HERMES produces a counter-example. It is a proof of \((R^\alpha, I_0^\alpha) \not\models \text{Secret}(S^\alpha)\).

### Positive verdict
HERMES generates a checkable proof based on the principle of induction for pre-reachability analysis.

Usage conditions of the protocol:
- \(S^\alpha \subseteq S_{\text{HERMES}}^\alpha\)
- \(S_{\text{HERMES}}^\alpha \cap I_0^\alpha = \emptyset\)

Correctness of HERMES:
- \(\text{pre}(R^\alpha, S_{\text{HERMES}}^\alpha) \subseteq S_{\text{HERMES}}^\alpha\)
- \(\text{pre}^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \equiv R^\alpha^*(I_0^\alpha) \cap S^\alpha = \emptyset\)
Certifying the abstract verdict of HERMES

Does the abstract system \((R^\alpha, I_0^\alpha)\) satisfies a property \(Secret(S^\alpha)\)?

**Negative verdict**
HERMES produces a counter-example. It is a proof of
\((R^\alpha, I_0^\alpha) \nsubseteq Secret(S^\alpha)\).

**Positive verdict**
HERMES generates a checkable proof based on the principle of induction for pre-reachability analysis

Usage conditions of the protocol

\[ S^\alpha \subseteq S^\alpha_{HERMES} \]
\[ S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset \]

Correctness of HERMES

\[ pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \]
\[ pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \equiv R^{\alpha^*}(I_0^\alpha) \cap S^\alpha = \emptyset \]
Abstract proof of stability

For each transition \( \tau \) of \( R^\alpha \), for each secret \( s \) in \( \text{pre}(\tau, S^\alpha_{HERMES}) \), we have to prove \( s \in S^\alpha_{HERMES} \).

Run an instrumented version of HERMES on \( S^\alpha_{HERMES} \)

All the attempt to add a new secret \( s \) fails, it already belongs to \( S^\alpha_{HERMES} \). The execution path of HERMES that leads to that conclusion gives the argument for proving \( s \in S^\alpha_{HERMES} \).

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.
Abstract proof of stability

For each transition $\tau$ of $R^\alpha$, for each secret $s$ in $\text{pre}(\tau, S^\alpha_{\text{HERMES}})$, we have to prove $s \in S^\alpha_{\text{HERMES}}$.

Run an instrumented version of HERMES on $S^\alpha_{\text{HERMES}}$

All the attempts to add a new secret $s$ fail, it already belongs to $S^\alpha_{\text{HERMES}}$. The execution path of HERMES that leads to that conclusion gives the argument for proving $s \in S^\alpha_{\text{HERMES}}$.

HERMES computations drives the proof

- Proof of a disjunction: The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- Proof of a existential property: requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.
Abstract proof of stability

For each transition $\tau$ of $R^\alpha$, for each secret $s$ in $\text{pre}(\tau, S^\alpha_{\text{HERMES}})$, we have to prove $s \in S^\alpha_{\text{HERMES}}$.

Run an instrumented version of HERMES on $S^\alpha_{\text{HERMES}}$

All the attempt to add a new secret $s$ fails, it already belongs to $S^\alpha_{\text{HERMES}}$. The execution path of HERMES that leads to that conclusion gives the argument for proving $s \in S^\alpha_{\text{HERMES}}$.

HERMES computations drives the proof

- Proof of a disjunction: The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- Proof of a existential property: requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.
Abstract proof of stability

For each transition $\tau$ of $R^\alpha$, for each secret $s$ in $\text{pre}(\tau, S^\alpha_{\text{HERMES}})$, we have to prove $s \in S^\alpha_{\text{HERMES}}$.

**Run an instrumented version of HERMES on $S^\alpha_{\text{HERMES}}$**

All the attempt to add a new secret $s$ fails, it already belongs to $S^\alpha_{\text{HERMES}}$. The execution path of HERMES that leads to that conclusion gives the argument for proving $s \in S^\alpha_{\text{HERMES}}$.

**HERMES computations drives the proof**

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.
Abstract proof of stability

For each transition $\tau$ of $R^\alpha$, for each secret $s$ in $\text{pre}(\tau, S^\alpha_{\text{HERMES}})$, we have to prove $s \in S^\alpha_{\text{HERMES}}$.

Run an instrumented version of HERMES on $S^\alpha_{\text{HERMES}}$

All the attempt to add a new secret $s$ fails, it already belongs to $S^\alpha_{\text{HERMES}}$. The execution path of HERMES that leads to that conclusion gives the argument for proving $s \in S^\alpha_{\text{HERMES}}$.

HERMES computations drives the proof

- **Proof of a disjunction**: The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.

- **Proof of a existential property**: requires to exhibit a witness of the property. HERMES records them.

- The reminder of the proof is managed by general proof-tactics.
Abstract proof of stability

For each transition $\tau$ of $R^\alpha$, for each secret $s$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $s \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on $S^\alpha_{HERMES}$

All the attempt to add a new secret $s$ fails, it already belongs to $S^\alpha_{HERMES}$. The execution path of HERMES that leads to that conclusion gives the argument for proving $s \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction**: The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.

- **Proof of a existential property**: requires to exhibit a witness of the property. HERMES records them.

- The reminder of the proof is managed by general proof-tactics.
Proof based on safeness of the abstraction

\[(R, S_{HERMES}), \alpha, (R^\alpha, S^\alpha_{HERMES})\]

\[\nabla^\alpha \pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}\]

\[\pre(R, S_{HERMES}) \subseteq S_{HERMES}\]

(PO$_1$) correct computation of the abstract image of $R, S_{HERMES}$

(PO$_2$) safeness of the abstraction $\alpha$

\[\pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \Rightarrow \pre(R, S_{HERMES}) \subseteq S_{HERMES}\]
Proof based on safeness of the abstraction

\[
(R, S_{HERMES}), \alpha, (R^\alpha, S^\alpha_{HERMES})
\]

\[
\Delta \alpha
\]

\[
\alpha(R, S_{HERMES}) = (R^\alpha, S^\alpha_{HERMES})
\]

\[
pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}
\]

\[
pre(R, S_{HERMES}) \subseteq S_{HERMES}
\]

(P\text{O}_1) correct computation of the abstract image of \( R, S_{HERMES} \)

(P\text{O}_2) safeness of the abstraction \( \alpha \)

\[
pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \Rightarrow pre(R, S_{HERMES}) \subseteq S_{HERMES}
\]
Proof based on safeness of the abstraction

\[ \alpha(R, S_{HERMES}) = (R^\alpha, S^\alpha_{HERMES}) \]

\((PO_1)\) correct computation of the abstract image of \(R, S_{HERMES}\)

\((PO_2)\) safeness of the abstraction \(\alpha\)

\[ pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \Rightarrow pre(R, S_{HERMES}) \subseteq S_{HERMES} \]
Concretization of the $\nabla^{\alpha}$ proof provided by HERMES

It avoids proving safeness of the abstraction by **removing** all references to the abstraction relation and abstract domains **from the proof**

Principle of the concretization function $[.] : \nabla \rightarrow \nabla$

- Abstract constants are replaced by constant symbols $c$ with additional hypothesis: $[C^{\alpha}] \sim c$ such that $\alpha(c, C^{\alpha})$

- Abstract equalities are equivalence relation on the concrete domain: $[\equiv^{\alpha}] \sim \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^{\alpha} \alpha(x, C^{\alpha}) \land \alpha(y, C^{\alpha})$

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$: $[P^{\alpha}] \sim \sim$-uniform version of $P$. 

R. Janvier, Y. Lakhnech, M. Périsn

Certification of Cryptographic Protocols
Concretization of the $\nabla^\alpha$ proof provided by HERMES

It avoids proving safeness of the abstraction by removing all references to the abstraction relation and abstract domains from the proof.

Principle of the concretization function $[\cdot] : \nabla \rightarrow \nabla$

- Abstract constants are replaced by constant symbols $c$ with additional hypothesis: $[C^\alpha] \sim c$ such that $\alpha(c, C^\alpha)$

- Abstract equalities are equivalence relation on the concrete domain: $[=^\alpha] \sim \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$ $[P^\alpha] \sim \sim$-uniform version of $P$
<table>
<thead>
<tr>
<th>Concretization of the $\nabla^\alpha$ proof provided by HERMES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>It avoids proving safeness of the abstraction</strong></td>
</tr>
<tr>
<td>by <em>removing</em> all references to the abstraction relation and</td>
</tr>
<tr>
<td>abstract domains <em>from the proof</em></td>
</tr>
<tr>
<td><strong>Principle of the concretization function $\lfloor . \rfloor : \nabla \rightarrow \nabla$</strong></td>
</tr>
<tr>
<td>• Abstract constants are replaced by <strong>constant symbols</strong> $c$ with additional hypothesis: $\lfloor C^\alpha \rfloor \leadsto c$ such that $\alpha(c, C^\alpha)$</td>
</tr>
<tr>
<td>• Abstract equalities are <strong>equivalence relation</strong> on the concrete domain: $\lfloor =^\alpha \rfloor \leadsto \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$</td>
</tr>
<tr>
<td>• Abstract predicates are replaced by <strong>concrete predicates with uniform valuation</strong> on the equivalence classes of $\sim$ $\lfloor P^\alpha \rfloor \leadsto \sim$-uniform version of $P$</td>
</tr>
</tbody>
</table>
Concretization of the $\nabla^\alpha$ proof provided by HERMES

It avoids proving safeness of the abstraction by removing all references to the abstraction relation and abstract domains from the proof.

Principle of the concretization function $[.] : \nabla \rightarrow \nabla$

- Abstract constants are replaced by constant symbols $c$ with additional hypothesis: $[C^\alpha] \leadsto c$ such that $\alpha(c, C^\alpha)$

- Abstract equalities are equivalence relation on the concrete domain: $[=^\alpha] \leadsto \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$ $[P^\alpha] \leadsto \sim$-uniform version of $P$
Concretization of the $\nabla^\alpha$ proof provided by HERMES

It avoids proving safeness of the abstraction by removing all references to the abstraction relation and abstract domains from the proof.

### Principle of the concretization function $\llbracket . \rrbracket : \nabla \rightarrow \nabla$

- Abstract constants are replaced by constant symbols $c$ with additional hypothesis: $\llbracket C^\alpha \rrbracket \leadsto c$ such that $\alpha(c, C^\alpha)$

- Abstract equalities are equivalence relation on the concrete domain: $\llbracket =^\alpha \rrbracket \leadsto \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$ $\llbracket P^\alpha \rrbracket \leadsto \sim\text{-uniform version of } P$
Concretization of the $\nabla^\alpha$ proof provided by HERMES

It avoids proving safeness of the abstraction by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

**Principle of the concretization function $[.] : \nabla \rightarrow \nabla$**

- Abstract constants are replaced by **constant symbols** $c$ with additional hypothesis: $[C^\alpha] \leadsto c$ such that $\alpha(c, C^\alpha)$

- Abstract equalities are **equivalence relation** on the concrete domain: $[=^\alpha] \leadsto \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$ $[P^\alpha] \leadsto \sim$-uniform version of $P$
Concretization of the $\nabla^\alpha$ proof provided by HERMES

It avoids proving safeness of the abstraction by removing all references to the abstraction relation and abstract domains from the proof.

Principle of the concretization function $[.] : \nabla \rightarrow \nabla$

- Abstract constants are replaced by constant symbols $c$ with additional hypothesis: $[C^\alpha] \rightsquigarrow c$ such that $\alpha(c, C^\alpha)$
  - concrete predicate

- Abstract equalities are equivalence relation on the concrete domain: $[=^\alpha] \rightsquigarrow \sim$ with $x \sim y \overset{\text{def}}{=} \exists C^\alpha \alpha(x, C^\alpha) \land \alpha(y, C^\alpha)$
  - concrete predicate

- Abstract predicates are replaced by concrete predicates with uniform valuation on the equivalence classes of $\sim$
  $[P^\alpha] \rightsquigarrow \sim$-uniform version of $P$
Uniform valuation of a predicate on equivalence classes

\[ P^-(x) \overset{\text{def}}{=} \forall y, y \sim x \Rightarrow P(y) \]
\[ P^+(x) \overset{\text{def}}{=} \exists y, y \sim x \land P(y) \]

**Remark:** \( P \Rightarrow P^+ \)
Uniform valuation of a predicate on equivalence classes

\[ P^-(x) \overset{\text{def}}{=} \forall y, y \sim x \Rightarrow P(y) \]

\[ P^+(x) \overset{\text{def}}{=} \exists y, y \sim x \land P(y) \]

uniform-restriction of \( P \)  
uniform-extension of \( P \)

Remark: \( P \Rightarrow P^+ \)
Uniform valuation of a predicate on equivalence classes

uniform-restriction of $P$

$$P^-(x) \overset{\text{def}}{=} \forall y, \ y \sim x \Rightarrow P(y)$$

concrete predicate

uniform-extension of $P$

$$P^+(x) \overset{\text{def}}{=} \exists y, \ y \sim x \land P(y)$$

concrete predicate

Remark: $P \Rightarrow P^+$
Proof by concretization

\[ R, S_{HERMES} = R, [\text{def. } S^\alpha_{HERMES}] \]

easy but not fully automatic

\[ R \Rightarrow \text{expanded def. } R^+ \]

\[ [\text{def. } R^\alpha], [\text{def. } S^\alpha_{HERMES}] = \text{expanded def. } R^+, [\text{def. } S^\alpha_{HERMES}] \]

trivial and automatic for \((R \Rightarrow R^+)\)

\[ pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \]

\[ [\text{def. } R^\alpha], [\text{def. } S^\alpha_{HERMES}] = R^+(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \]

\[ pre(R, S_{HERMES}) \subseteq S_{HERMES} \]

\[ R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \]
Proof by concretization

\[ R, S_{\text{HERMES}} ] = [\text{def. } R^\alpha], [\text{def. } S^\alpha_{\text{HERMES}}] = \text{expanded def. } R^+, [\text{def. } S^\alpha_{\text{HERMES}}] \]

\[ \text{easy but not fully automatic} \]

\[ R \xrightarrow{\alpha} \text{expanded def. } R^+ \]

\[ [\text{def. } R^\alpha] = [\text{def. } S^\alpha_{\text{HERMES}}] = R_? \Rightarrow \text{expanded def. } R^+, [\text{def. } S^\alpha_{\text{HERMES}}] \]

\[ \text{trivial and automatic for } (R \Rightarrow R^+) \]

\[ \text{pre}(R, S_{\text{HERMES}}) \subseteq S_{\text{HERMES}} \]

\[ \text{pre}(R, S_{\text{HERMES}}) \subseteq S_{\text{HERMES}} \]

R. Janvier, Y. Lakhnech, M. Périn

Certification of Cryptographic Protocols
Proof by concretization

\[
R, \quad S^{\alpha}_{HERMES} = R \quad \text{easy but not fully automatic}
\]

\[
\text{expanded def. } R^+, \quad \text{def. } S^{\alpha}_{HERMES}
\]

\[
\text{trivial and automatic for } (R \Rightarrow R^+)
\]

\[
\text{expanded def. } R^+, \quad \text{def. } S^{\alpha}_{HERMES}
\]

\[
\text{expanded def. } R^+, \quad \text{def. } S^{\alpha}_{HERMES}
\]
Proof by concretization

\[
R, \ S_{HERMES} = R \quad \text{[def. } S_{HERMES}^{\alpha}] \\
\text{(PO}_1) \\
\text{easy but not fully automatic} \\
\text{R } \Rightarrow \text{ expanded def. } R^+ \\
\text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\therefore \ \text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R^{\alpha}, S_{HERMES}^{\alpha}) \subseteq S_{HERMES}^{\alpha}] = R^+(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{(PO}_2) \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{trivial and automatic for } (R \Rightarrow R^+) \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
\text{[pre(} R, S_{HERMES}) \subseteq S_{HERMES}] = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \\
\text{[def. } R^{\alpha}], \ [\text{def. } S_{HERMES}^{\alpha}] = \text{expanded def. } R^+, \ [\text{def. } S_{HERMES}^{\alpha}] \\
Proof by concretization

\[ R, S_{HERMES} = R, \quad \text{[def. } S^\alpha_{HERMES}\text{]} \]

\begin{equation}
\begin{aligned}
(PO_1) & \\
\text{easy but not fully automatic} & \\
\text{expanded def. } R^+ & \\
\text{[def. } R^\alpha\text{], [def. } S^\alpha_{HERMES}\text{]} = \text{expanded def. } R^+, \text{[def. } S^\alpha_{HERMES}\text{]} & \\

\begin{equation}
\begin{aligned}
[\nabla^\alpha] & \\
\left[ \text{pre}(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} \right] & = R^+ (s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} & \\
(PO_2) & \\
\text{trivial and automatic for } (R \Rightarrow R^+) & \\
\text{pre}(R, S_{HERMES}) \subseteq S_{HERMES} & = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}
\end{aligned}
\end{equation}
\end{aligned}
\end{equation}
Proof by concretization

\[ R, \ S_{HERMES} = R \quad \text{easy but not fully automatic} \]

\[ (PO_1) \]

\[ \text{expanded def. } R^+ \]

\[ \text{def. } R^\alpha, \ \text{def. } S^\alpha_{HERMES} = \text{expanded def. } R^+, \ \text{def. } S^\alpha_{HERMES} \]

\[ \pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES} = R^+(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \]

\[ (PO_2) \]

\[ \pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s, s') \land s' \in S_{HERMES} \Rightarrow s \in S_{HERMES} \]
Proof by concretization

\[ \begin{align*}
R, \ S_{\text{HERMES}} &= R, \ [\text{def. } S^\alpha_{\text{HERMES}}] \\
(PO_1) &\quad \text{easy but not fully automatic}
\end{align*} \]

\[ \begin{align*}
[\text{def. } R^\alpha], \ [\text{def. } S^\alpha_{\text{HERMES}}] &= \text{expanded def. } R^+, \ [\text{def. } S^\alpha_{\text{HERMES}}] \\
\triangledown^\alpha &\quad \text{trivial and automatic for } (R \Rightarrow R^+) \\
(PO_2)
\end{align*} \]

\[ \begin{align*}
\text{pre}(R^\alpha, S^\alpha_{\text{HERMES}}) \subseteq S^\alpha_{\text{HERMES}} &= R^+(s, s') \land s' \in S_{\text{HERMES}} \Rightarrow s \in S_{\text{HERMES}} \\
\text{pre}(R, S_{\text{HERMES}}) \subseteq S_{\text{HERMES}} &= R(s, s') \land s' \in S_{\text{HERMES}} \Rightarrow s \in S_{\text{HERMES}}
\end{align*} \]


**Certification versus Proof-Carrying Code**

- **Challenge of PCC:** redacting the size of proof and complexity of proof-checking (not so important for certification)

- **Challenge of certification:** reducing the Trusted Computing Base to the proof-checker
  
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB

- Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through
  
  - the Verification Condition Generator in PCC: $\mathrm{VCG}(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)
Certification versus Proof-Carrying Code

- **Challenge of PCC:** reducing the size of proof and complexity of proof-checking (not so important for certification)

- **Challenge of certification:** reducing the Trusted Computing Base to the proof-checker
  - HERMES’s uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB

Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through
  - the Verification Condition Generator in PCC: $\text{VCG}(S, \varphi)$
  - extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Certification versus Proof-Carrying Code

- **Challenge of PCC:** *reducing the size of proof and complexity of proof-checking* (not so important for certification)

- **Challenge of certification:** *reducing the Trusted Computing Base to the proof-checker*
  - HERMES’s uses the Coq-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
  - Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through
    - the Verification Condition Generator in PCC: $\text{VCG}(S, \varphi)$
    - extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Certification versus Proof-Carrying Code

- **Challenge of PCC**: reducing the size of proof and complexity of proof-checking (not so important for certification)

- **Challenge of certification**: reducing the Trusted Computing Base to the proof-checker

  - HERMES’s uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB

Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through

- the Verification Condition Generator in PCC: $\text{VCG}(S, \varphi)$
- extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Certification versus Proof-Carrying Code

- **Challenge of PCC:** reducing the size of proof and complexity of proof-checking (not so important for certification)

- **Challenge of certification:** reducing the Trusted Computing Base to the proof-checker
  
  - HERMES’s uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB

- Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through
  
  - the Verification Condition Generator in PCC: $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Challenge of PCC: reducing the size of proof and complexity of proof-checking (not so important for certification)

Challenge of certification: reducing the Trusted Computing Base to the proof-checker

- HERMES’s uses the COQ-engine where proof-checking is type checking
- PPC places the VCG to the TCB

Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through

- the Verification Condition Generator in PCC: $VCG(S, \varphi)$
- extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Challenge of PCC: reducing the size of proof and complexity of proof-checking (not so important for certification)

Challenge of certification: reducing the Trusted Computing Base to the proof-checker

- HERMES’s uses the COQ-engine where proof-checking is type checking
- PPC places the VCG to the TCB

Both establish a checkable relation between the system $S$, the property $\varphi$ and the proof $\nabla$ through

- the Verification Condition Generator in PCC: $VCG(S, \varphi)$
- extraction of the protocol and the property from HERMES’s proof (No need for VCG)
Related work

1. **BLAST [T.Henziger et al., 2002]**
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. **[K.Namjoshi, 2003]**
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. **[Many people in MC community]**
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   - generation of concrete proof of mu-calculus properties
     - in a specific deduction system
     - level of automatization?

3. [Many people in MC community]
   - generation of checkable witness of the computations of a model-checker
     - requires to trust the checker
     - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. **BLAST [T. Henziger et al., 2002]**
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. **[K. Namjoshi, 2003]**
   - generation of concrete proof of mu-calculus properties
     - in a specific deduction system
     - level of automatization?

3. **[Many people in MC community]**
   - generation of checkable witness of the computations of a model-checker
     - requires to trust the checker
     - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of μ-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. **BLAST** [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. **[K. Namjoshi, 2003]**
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. **[Many people in MC community]**
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
Related work

1. BLAST [T. Henziger et al., 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG

2. [K. Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?

3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?
HERMES is available on-line

http://www.verimag.imag.fr/ → DCS → software → HERMES