Stateful Protocol Verification in AIF-ω, a Tutorial

Alessandro Bruni

joint work with Sebastian Mödersheim

material available at http://alessandrobruni.name

LAMAS SING 2016
Roadmap

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
Importance of Security Protocols

security protocols run the internet, banking, and now even more of our interconnected physical world (ie smart homes, cars, devices) without them we would be vulnerable to remote attackers

Examples
Need to Verify Security Protocols

- Designing and implementing secure protocols is hard and prone to subtle mistakes
Need to Verify Security Protocols

- Designing and implementing secure protocols is hard and prone to subtle mistakes
- Example:
  - Needham-Schröder session key establishment, proposed in 1978
  - man-in-the-middle attack found by Lowe in 1995
Need to Verify Security Protocols

- Designing and implementing secure protocols is hard and prone to subtle mistakes
- Example:
  - Needham-Schröder session key establishment, proposed in 1978
  - man-in-the-middle attack found by Lowe in 1995
  - verified using BAN logic (also broken)
Need to Verify Security Protocols

- Designing and implementing secure protocols is hard and prone to subtle mistakes
- Example:
  - Needham-Schröder session key establishment, proposed in 1978
  - man-in-the-middle attack found by Lowe in 1995
  - verified using BAN logic (also broken)
- Morale: design mistakes can go unnoticed for years, formal reasoning can help to discover them
Need to Verify Security Protocols

- Designing and implementing secure protocols is hard and prone to subtle mistakes
- Example:
  - Needham-Schröder session key establishment, proposed in 1978
  - man-in-the-middle attack found by Lowe in 1995
  - verified using BAN logic (also broken)
- Morale: design mistakes can go unnoticed for years, formal reasoning can help to discover them

- Two main formal approaches:
  - symbolic reasoning (Dolev-Yao)
  - computational reasoning
Dolev-Yao representation of security protocols

- Security primitives as black-boxes
- Symbolic representation of crypto operations:
  \[ senc(M, K), \ sdec(C, K), \ aenc(M, Pk), \ adec(C, Sk), \ pk(A), \ sk(A), \ sign(M, Sk), \ check(S, Pk), \ldots \]
- Equational theory \( (\equiv_E) \):
  \[ sdec(senc(M, K), K) \equiv_E M, \]
  \[ adec(aenc(M, pk(A)), sk(A)) \equiv_E M, \]
  \[ check(sign(M, Sk), pk(Sk)) \equiv_E ok \]
  all other behaviours are impossible.
- Protocol represented by inference rules: given necessary inputs, the expected outputs are produced
- Attacker controls the channel:
  can construct, inject, delete and eavesdrop messages
Dolev-Yao representation of security protocols (2)

Interesting properties

- **secrecy**: a certain piece of information is not derivable given the rules
- **authentication**: the information exchanged within the protocol is authentic
  - no single definition: authentic source, freshness, recentness, correspondences
- **indistinguishability**: the attacker cannot distinguish between two sessions of the protocol
  - e.g.: voting systems
Dolev-Yao representation of security protocols (3)

Advantages

- Simple symbolic representation
- Good for deductive reasoning
- Automatic tool support
  ProVerif, SPASS, OFMC, SATMC, AIF-ω and many others

Disadvantages

- Misses some potential problems: e.g., possible to find hash collisions, known-plaintext attacks, type-flaw attacks etc.
- Probabilistic crypto out of reach: fashionable example: blockchain
- Undecidable in general: fresh values, unbounded sessions, unbounded agents, undecidable equational theories
Dolev-Yao representation of security protocols (3)

Advantages

- Simple symbolic representation
- Good for deductive reasoning
- Automatic tool support
  ProVerif, SPASS, OFMC, SATMC, AIF-ω and many others

Disadvantages

- Misses some potential problems:
  e.g.: possible to find hash collisions, known-plaintext attacks, type-flaw attacks etc.
Dolev-Yao representation of security protocols (3)

Advantages

▶ Simple symbolic representation
▶ Good for deductive reasoning
▶ Automatic tool support
  ProVerif, SPASS, OFMC, SATMC, AIF-ω and many others

Disadvantages

▶ Misses some potential problems:
  e.g.: possible to find hash collisions, known-plaintext attacks,
  type-flaw attacks etc.
▶ Probabilistic crypto out of reach
  fashionable example: blockchain
Dolev-Yao representation of security protocols (3)

Advantages

▶ Simple symbolic representation
▶ Good for deductive reasoning
▶ Automatic tool support
  ProVerif, SPASS, OFMC, SATMC, AIF-ω and many others

Disadvantages

▶ Misses some potential problems:
  e.g.: possible to find hash collisions, known-plaintext attacks,
  type-flaw attacks etc.
▶ Probabilistic crypto out of reach
  fashionable example: blockchain
▶ Undecidable in general:
  fresh values, unbounded sessions, unbounded agents, undecidable
  equational theories
Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
Needham–Schröder, our Drosophila

Protocol:

\[ A \to B : \{N_A, A\}_{pk(B)} \]  \hspace{1cm} (1)
\[ B \to A : \{N_A, N_B\}_{pk(A)} \]  \hspace{1cm} (2)
\[ A \to B : \{N_B\}_{pk(B)} \]  \hspace{1cm} (3)

[AIF-ω model]
Needham–Schröder, our Drosophila

Protocol:

\[ A \rightarrow B : \{N_A, A\}_{pk(B)} \] (1)

\[ B \rightarrow A : \{N_A, N_B\}_{pk(A)} \] (2)

\[ A \rightarrow B : \{N_B\}_{pk(B)} \] (3)

Deduction rules:

\[ \frac{\text{aenc}(\text{pair}(N_A, A), pk(B))}{\text{aenc}(\text{pair}(N_A, A), pk(B))} \] (1)

\[ \frac{\text{aenc}(\text{pair}(N_A, A), pk(B))}{\text{aenc}(\text{pair}(N_A, N_B), pk(A))} \] (2)

\[ \frac{\text{aenc}(\text{pair}(N_A, N_B), pk(B))}{\text{aenc}(N_B, pk(A))} \] (3)

[AIF-ω model]
Needham–Schröder–Lowe (fixed)

Protocol:

\[ A \rightarrow B : \{N_A, A\}_{pk(B)} \]  \hspace{1cm}  (1)
\[ B \rightarrow A : \{N_A, N_B\}_{pk(A)} \]  \hspace{1cm}  (2)
\[ A \rightarrow B : \{N_B\}_{pk(B)} \]  \hspace{1cm}  (3)

Attack trace:

\[ A \rightarrow I : \{N_A, A\}_{pk(I)} \]
\[ I \rightarrow B : \{N_A, A\}_{pk(B)} \]
\[ B \rightarrow I : \{N_A, N_B\}_{pk(A)} \]
\[ I \rightarrow A : \{N_A, N_B\}_{pk(A)} \]
\[ A \rightarrow I : \{N_B\}_{pk(I)} \]
\[ I \rightarrow B : \{N_B\}_{pk(B)} \]

[AIF-ω model]
Needham–Schröder–Lowe (fixed)

Protocol:

\[ A \rightarrow B : \{ N_A, A \}_{pk(B)} \]  \hspace{1cm} (1)

\[ B \rightarrow A : \{ N_A, N_B, B \}_{pk(A)} \]  \hspace{1cm} (2)

\[ A \rightarrow B : \{ N_B \}_{pk(B)} \]  \hspace{1cm} (3)

[AIF-\omega model]
Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
Motivation for AIF-ω

Horn-clause representation of security protocols

- successful verification method (e.g. ProVerif, SPASS)
- abstraction of state, sessions and freshness
- **problem**: hard to verify systems with state, e.g. key revocation protocols, TPM, timestamps, databases (e.g. web shops)...

Allowing for stateful protocols without destroying the benefits of the ProVerif method

▶ AIF, StatVerif, SetPi: annotate the usual predicates (e.g. $i(·)$) with terms representing the current state.

▶ However, the state information in all the stateful approaches must be limited to a fixed size for termination.

For StatVerif and AIF this in particular means a limitation to a fixed number of agents (but unbounded sessions)

Research question:

How can we verify protocols with unbounded principals, each with their own persistent state?
Motivation for AIF-ω

Horn-clause representation of security protocols

- successful verification method (e.g. ProVerif, SPASS)
- abstraction of state, sessions and freshness
- **problem**: hard to verify systems with state, e.g. key revocation protocols, TPM, timestamps, databases (e.g. web shops)...

Allowing for stateful protocols without destroying the benefits of the ProVerif method

- AIF, StatVerif, SetPi: annotate the usual predicates (e.g. \( i(\cdot) \)) with terms representing the current state.
- **However**, the state information in all the stateful approaches must be limited to a fixed size for termination. For StatVerif and AIF this in particular means a limitation to a fixed number of agents (but unbounded sessions)
Motivation for AIF-ω

Horn-clause representation of security protocols

- successful verification method (e.g. ProVerif, SPASS)
- abstraction of state, sessions and freshness
- **problem**: hard to verify systems with state, e.g. key revocation protocols, TPM, timestamps, databases (e.g. web shops)...

Allowing for stateful protocols without destroying the benefits of the ProVerif method

- AIF, StatVerif, SetPi: annotate the usual predicates (e.g. $i(\cdot)$) with terms representing the current state.
- **However**, the state information in all the stateful approaches must be limited to a fixed size for termination. For StatVerif and AIF this in particular means a limitation to a fixed number of agents (but unbounded sessions)

**Research question:**

How can we verify protocols with unbounded principals, each with their own persistent state?
Contributions

- A language, AIF-ω, with support for the verification of stateful protocols with unbounded principals
  - state represented by countably infinite sets (databases) indexed into finite families
- Soundness proof of our translation
- Implementation:
  - translator from AIF-ω into Horn clauses, ProVerif and SPASS used as solvers.
- Case studies
Set-Membership Abstraction: Key Ideas

- **Organize data into countable families of sets (data-bases), indexed by agent names.**
  
  Example: user $a$ has a set $\text{ring}(a)$ of current keys, registered at the key server $s_1$, either in $\text{valid}(s_1, a)$ or in $\text{revoked}(s_1, a)$. 

Set-Membership Abstraction: Key Ideas

- Organize data into countable families of sets (data-bases), indexed by agent names.
  Example: user $a$ has a set $\text{ring}(a)$ of current keys, registered at the key server $s_1$, either in $\text{valid}(s_1, a)$ or in $\text{revoked}(s_1, a)$.

- Disjointness assumption (for a fixed representation).
  All sets of the same family are pairwise disjoint.
  E.g. $\forall A, B \in \text{User} . A \neq B \implies \text{ring}(A) \cap \text{ring}(B) = \emptyset$.
  Violations of the assumption constitute an attack.
Set-Membership Abstraction: Key Ideas

- **Organize data into countable families of sets (data-bases), indexed by agent names.**
  Example: user $a$ has a set $\text{ring}(a)$ of current keys, registered at the key server $s_1$, either in $\text{valid}(s_1, a)$ or in $\text{revoked}(s_1, a)$.

- **Disjointness assumption (for a fixed representation).**
  All sets of the same family are pairwise disjoint.
  E.g. $\forall A, B \in \text{User} . A \neq B \implies \text{ring}(A) \cap \text{ring}(B) = \emptyset$.
  Violations of the assumption constitute an attack.

- **Annotate the current state of an object with a term.**
  Given the ordering $\langle \text{ring}, \text{valid}, \text{revoked} \rangle$, a key $pk$ satisfying only $pk \in \text{valid}(s_1, a)$ is annotated by $\langle 0, \text{valid}(s_1, a), 0 \rangle$. 
Set-Membership Abstraction: Key Ideas

- **Organize data into countable families of sets (data-bases), indexed by agent names.**
  Example: user \( a \) has a set \( \text{ring}(a) \) of current keys, registered at the key server \( s_1 \), either in \( \text{valid}(s_1, a) \) or in \( \text{revoked}(s_1, a) \).

- **Disjointness assumption (for a fixed representation).**
  All sets of the same family are pairwise disjoint.
  E.g. \( \forall A, B \in \text{User} . A \neq B \implies \text{ring}(A) \cap \text{ring}(B) = \emptyset \).
  Violations of the assumption constitute an attack.

- **Annotate the current state of an object with a term.**
  Given the ordering \( \langle \text{ring}, \text{valid}, \text{revoked} \rangle \), a key \( pk \) satisfying only \( pk \in \text{valid}(s_1, a) \) is annotated by \( \langle 0, \text{valid}(s_1, a), 0 \rangle \).

- **Term implications: representing set-membership changes**
  If \( pk \circ \langle 0, \text{valid}(s_1, a), 0 \rangle \impliedby pk \circ \langle 0, 0, \text{revoked}(s_1, a) \rangle \),
  then for every context \( C[::] \) where \( C[s] \) holds, also \( C[t] \) holds.
Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
Hello World: a Hardware Security Module

- Hardware token, generates a secret $s$
- Reveals either the left projection $\text{left}(s)$ or the right projection $\text{right}(s)$
- Attacker should not learn both $\text{left}(s)$ and $\text{right}(s)$ at the same time.

[AIF-\omega model]
AIF  key revocation protocol [AIF-ω model]

Types:

\[
\begin{align*}
\text{Honest} & \quad = \quad \{a, b, c\}; \\
\text{Dishon} & \quad = \quad \{i\}; \\
\text{User} & \quad = \quad \text{Honest} \cup \text{Dishon}; \\
\text{Server} & \quad = \quad \{s_1, s_2\};
\end{align*}
\]

Sets:

\[
\begin{align*}
\text{ring(} \text{User} \text{)}, \quad \text{valid(} \text{Server}, \text{User} \text{)}, \quad \text{revoked(} \text{Server}, \text{User} \text{)};
\end{align*}
\]
AIF key revocation protocol [AIF-ω model]

Types:

\[ Honest = \{a, b, c\}; \quad Dishon = \{i\}; \]
\[ User = Honest \cup Dishon; \quad Server = \{s_1, s_2\}; \]

Sets:

\[ ring(User), \quad valid(Server, User), \quad revoked(Server, User); \]

Rules:

\[ registerOutOfBand(U : User, S : Server) = \]
\[ \equiv[PK] \Rightarrow PK \in ring(U) \cdot PK \in valid(S, U) \cdot i(PK) \]
AIF key revocation protocol [AIF-ω model]

Types:

\[\text{Honest} = \{a, b, c\}; \quad \text{Dishon} = \{i\};\]
\[\text{User} = \text{Honest} \cup \text{Dishon}; \quad \text{Server} = \{s_1, s_2\};\]

Sets:

\[\text{ring}(\text{User}), \ \text{valid}(\text{Server}, \text{User}), \ \text{revoked}(\text{Server}, \text{User});\]

Rules:

\[\text{registerOutOfBand}(U : \text{User}, S : \text{Server}) = \]
\[\Rightarrow [PK] \Rightarrow PK \in \text{ring}(U) \cdot PK \in \text{valid}(S, U) \cdot i(PK)\]

\[\text{updateKey}(U : \text{User}, S : \text{Server}, PK : \text{val}, NPK : \text{val}) = \]
\[i(\text{sign}_{\text{inv}(PK)}(S, U, NPK)) \cdot PK \in \text{valid}(S, U) \cdot NPK \notin \text{valid}(\_ , \_ ) \cdot NPK \notin \text{revoked}(\_ , \_ ) \Rightarrow PK \in \text{revoked}(S, U) \cdot NPK \in \text{valid}(S, U) \cdot i(\text{inv}(PK))\]
AIF key revocation protocol [AIF-ω model]

Types:

\[
\begin{align*}
\text{Honest} & \quad = \quad \{a, b, c\}; \\
\text{Dishon} & \quad = \quad \{i\}; \\
\text{User} & \quad = \quad \text{Honest} \cup \text{Dishon}; \\
\text{Server} & \quad = \quad \{s_1, s_2\};
\end{align*}
\]

Sets:

\[
\text{ring}(\text{User}), \quad \text{valid}(\text{Server}, \text{User}), \quad \text{revoked}(\text{Server}, \text{User});
\]

Rules:

\[
\begin{align*}
\text{registerOutOfBand}(U : \text{User}, S : \text{Server}) & = \\
= [PK] \Rightarrow PK \in \text{ring}(U) \cdot PK \in \text{valid}(S, U) \cdot \text{i}(PK)
\end{align*}
\]

\[
\begin{align*}
\text{updateKey}(U : \text{User}, S : \text{Server}, PK : \text{val}, NPK : \text{val}) & = \\
i(\text{sign}_{\text{inv}}(PK)(S, U, NPK)) \cdot PK \in \text{valid}(S, U) \cdot \\
NPK \notin \text{valid}(_, _) \cdot NPK \notin \text{revoked}(_, _) \Rightarrow PK \in \text{revoked}(S, U) \cdot NPK \in \text{valid}(S, U) \cdot \text{i}(\text{inv}(PK))
\end{align*}
\]

\[
\begin{align*}
\text{attackDef}(U : \text{Honest}, S : \text{Server}, PK : \text{val}) & = \\
i(\text{inv}(PK)) \cdot PK \in \text{valid}(S, U) \Rightarrow \text{attack}.
\end{align*}
\]
AIF-ω key revocation protocol [AIF-ω model]

Types:

\[
\begin{align*}
Honest & = \{a, b, c, \ldots\}; \\
Dishon & = \{i, \ldots\}; \\
User & = Honest \cup Dishon; \\
Server & = \{s_1, s_2, \ldots\};
\end{align*}
\]

Sets:

\[
ring(User), \ valid(Server, User), \ revoked(Server, User);
\]

Rules:

\[
\begin{align*}
\text{registerOutOfBand}(U : User, S : Server) = \\
&\Rightarrow PK \in ring(U) \cdot PK \in valid(S, U) \cdot i(PK) \\
\text{updateKey}(U : User, S : Server, PK : val, NPK : val) = \\
&i(\text{sign}_{\text{inv}(PK)}(S, U, NPK)) \cdot PK \in valid(S, U) \cdot NPK \notin valid(-, -) \cdot NPK \notin revoked(-, -) \\
&\Rightarrow PK \in revoked(S, U) \cdot NPK \in valid(S, U) \cdot i(\text{inv}(PK)) \\
\text{attackDef}(U : Honest, S : Server, PK : val) = \\
i(\text{inv}(PK)) \cdot PK \in valid(S, U) \Rightarrow \text{attack}.
\end{align*}
\]
updateKey\((U : User, S : Server, PK : val, NPK : val) = \]
\[
[\ldots] \cdot PK \in \text{valid}(S, U) \cdot NPK \notin \text{valid}(\_, \_ \_ \_) \cdot NPK \notin \text{revoked}(\_, \_ \_ \_)
\Rightarrow PK \in \text{revoked}(S, U) \cdot NPK \in \text{valid}(S, U) \cdot [\ldots]
\]

Before: \(pk@\langle 0, \text{valid}(s_1, a), 0 \rangle, npk@\langle \text{ring}(a), 0, 0 \rangle\)

\(\text{ring}(a)\)

\(\text{valid}(s_1, a)\)

\(\text{npk} \rightarrow \text{npk}\)

\(pk\)

\(\text{revoked}(s_1, a)\)

After: \(pk@\langle 0, 0, \text{revoked}(s_1, a) \rangle, npk@\langle \text{ring}(a), \text{valid}(s_1, a), 0 \rangle\)
updateKey\((s_1, a)\)

\[
\text{ring}(\text{User})
\]

\[
\text{valid}(\text{Server}, \text{User})
\]

\[
\text{revoked}(\text{Server}, \text{User})
\]

\[
npk @ \langle 0, \text{valid}(s_1, a), 0 \rangle \quad \rightarrow \quad pk @ \langle 0, 0, \text{revoked}(s_1, a) \rangle
\]

\[
npk @ \langle \text{ring}(a), 0, 0 \rangle \quad \rightarrow \quad npk @ \langle \text{ring}(a), \text{valid}(s_1, a), 0 \rangle
\]
updateKey\((s_1, a)\)
Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
Inadmissible (stupid) rules

- Rules violating *disjointness* on RHS:

  \[ r(X : \text{val}) = p(X) \Rightarrow X \in s_1(a) \]
Inadmissible (stupid) rules

- Rules violating disjointness on RHS:

\[ r(X : \text{val}) = p(X) \Rightarrow X \in s_1(a) \]

Can be compiled into:

\[ r_1(X : \text{val}) = p(X) \cdot X \notin s_1(\_ \_ ) \Rightarrow X \in s_1(a) \]
\[ r_2(X : \text{val}, A : T) = p(X) \cdot X \in s_1(A) \Rightarrow \text{attack} \]
Inadmissible (stupid) rules

- Rules violating *disjointness* on RHS:

\[ r(X : val) = p(X) \Rightarrow X \in s_1(a) \]

Can be compiled into:

\[ r_1(X: val) = p(X) \cdot X \notin s_1(_) \Rightarrow X \in s_1(a) \]
\[ r_2(X: val, A: T) = p(X) \cdot X \in s_1(A) \Rightarrow \text{attack} \]

- Rules violating *disjointness* on LHS:

\[ r(A : T, B : T, X : val) = p(X) \cdot X \in s_1(A) \cdot X \in s_1(B) \Rightarrow \ldots \]

only executes iff \( A = B \).
Translating $\textit{updateKey}(U, S)$ into Horn clauses

1. Compute equivalence class

\[
i(\text{sign}_{\text{inv}(PK)}(S, U, NPK)) \cdot \begin{array}{l}
PK \in \text{valid}(S, U) \cdot \\
NPK \notin \text{valid}(\_ , \_ ) \cdot NPK \notin \text{revoked}(\_ , \_ )
\end{array}
\]

\[
\Rightarrow
PK \in \text{revoked}(S, U) \cdot \\
NPK \in \text{valid}(S, U) \cdot \\
i(\text{inv}(PK))
\]
Translating \textit{updateKey}(U, S) into Horn clauses

1. Compute equivalence class
2. Substitute values with their class annotation

\[ i(\text{sign}_{\text{inv}}(\langle X_1, \text{valid}(S, U), X_2 \rangle))(S, U, \langle X_3, 0, 0 \rangle)) \cdot \]

\[ \begin{align*}
PK \in \text{valid}(S, U) \cdot \\
NPK \notin \text{valid}(\_, \_) \cdot NPK \notin \text{revoked}(\_, \_) \Rightarrow \\
PK \in \text{revoked}(S, U) \cdot \\
NPK \in \text{valid}(S, U) \cdot \\
i(\text{inv}(\langle X_1, 0, \text{revoked}(S, U) \rangle))
\end{align*} \]
Translating \textit{updateKey}(U, S) into Horn clauses

1. Compute equivalence class
2. Substitute values with their class annotation
3. Term implication predicates (state transitions)

\begin{align*}
i(\text{sign}_{\text{inv}}(\langle X_1, \text{valid}(S, U), X_2 \rangle))(S, U, \langle X_3, 0, 0 \rangle)) & \cdot \text{EquivalenceClass} \\
PK & \in \text{valid}(S, U) \cdot PK \quad X_1 \quad \text{valid}(S, U) \quad X_2 \\
NPK & \notin \text{valid}(~, ~) \cdot NPK \notin \text{revoked}(~, ~) \quad NPK \quad X_3 \quad 0 \quad 0 \\
\Rightarrow & \\
PK & \in \text{revoked}(S, U) \cdot PK \quad X_1 \quad 0 \quad \text{revoked}(S, U) \\
NPK & \in \text{valid}(S, U) \cdot NPK \quad X_3 \quad \text{valid}(S, U) \quad 0 \\
i(\text{inv}(\langle X_1, 0, \text{revoked}(S, U) \rangle)) & \\
\langle X_1, \text{valid}(S, U), 0 \rangle & \rightarrow \langle X_1, 0, \text{revoked}(S, U) \rangle \\
\langle X_3, 0, 0 \rangle & \rightarrow \langle X_3, \text{valid}(S, U), 0 \rangle
\end{align*}
Translating \textit{updateKey}(U, S) into Horn clauses

1. Compute equivalence class
2. Substitute values with their class annotation
3. Term implication predicates (state transitions)
4. State transition rules

\begin{align*}
i(\text{sign}_{\text{inv}}(\langle X_1, \text{valid}(S, U), X_2 \rangle))&(S, U, \langle X_3, 0, 0 \rangle)) \\
PK &\in \text{valid}(S, U) \\
NPK &\notin \text{valid}(\_, \_) \cdot NPK &\notin \text{revoked}(\_, \_) \\
\Rightarrow & \\
PK &\in \text{revoked}(S, U) \\
NPK &\in \text{valid}(S, U) \\
i(\text{inv}(\langle X_1, 0, \text{revoked}(S, U) \rangle))&(\langle X_1, \text{valid}(S, U), 0 \rangle \rightarrow \langle X_1, 0, \text{revoked}(S, U) \rangle) \\
\langle X_3, 0, 0 \rangle &\rightarrow \langle X_3, \text{valid}(S, U), 0 \rangle \\
\forall C[\cdot]. \langle X_3, 0, 0 \rangle &\rightarrow \langle X_3, \text{valid}(S, U), 0 \rangle \cdot C[\langle X_3, 0, 0 \rangle] \implies C[\langle X_3, \text{valid}(S, U), 0 \rangle] \\
\end{align*}
Translating \texttt{updateKey}$(U, S)$ into Horn clauses

1. Compute equivalence class
2. Substitute values with their class annotation
3. Term implication predicates (state transitions)
4. State transition rules
5. Quantify agents over their types

\[
\begin{align*}
\text{usr}(U) \cdot \text{srv}(S) & \\
\text{i(sign}_{\text{inv}}(\langle X_1, \text{valid}(S, U), X_2 \rangle))(S, U, \langle X_3, 0, 0 \rangle) & \\
PK \in \text{valid}(S, U) & \\
NPK \notin \text{valid}(\_, \_) \cdot \& \text{NPK} \notin \text{revoked}(\_, \_) & \\
\Rightarrow & \\
PK \in \text{revoked}(S, U) & \\
NPK \in \text{valid}(S, U) & \\
i(\text{inv}(\langle X_1, 0, \text{revoked}(S, U) \rangle)) & \\
\langle X_1, \text{valid}(S, U), 0 \rangle \rightarrow \langle X_1, 0, \text{revoked}(S, U) \rangle & \\
\langle X_3, 0, 0 \rangle \rightarrow \langle X_3, \text{valid}(S, U), 0 \rangle & \\
\forall C[\_] \cdot \langle X_3, 0, 0 \rangle \rightarrow \langle X_3, \text{valid}(S, U), 0 \rangle \cdot C[\langle X_3, 0, 0 \rangle] \Rightarrow C[\langle X_3, \text{valid}(S, U), 0 \rangle] & \\
\end{align*}
\]
Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions
## Experimental Results: Key Server Example

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>Backend</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Honest</strong></td>
<td><strong>Dishon</strong></td>
<td><strong>Server</strong></td>
<td><strong>ProVerif</strong></td>
</tr>
<tr>
<td>AIF</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.135s</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.418s</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2.057s</td>
</tr>
<tr>
<td>AIF-ω</td>
<td>ω</td>
<td>ω</td>
<td>ω</td>
</tr>
</tbody>
</table>
Conclusions

We extend a successful method in a successful way:

- The extension allows verification of stateful protocols with unbounded number of agents
- AIF-\(\omega\): clear specification language that allows exactly what the method can handle
- Soundness of the analysis for all AIF-\(\omega\) specifications
- Implementation using ProVerif and SPASS
- Case-studies:
  - PKCS11, SeVeCom, FuturEID, CANAuth
Thank you :)