Stateful Protocol Verification in AIF-ω, a Tutorial

Alessandro Bruni

joint work with Sebastian Mödersheim

material available at `http://alessandrobruni.name`

LAMAS SING 2016


www.waterlane.pl
SONICA I
GDAŃSK

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 phisical 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 unoticed 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 unoticed 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 ($=_E$):
  $sdec(senc(M, K), K) =_E M,$
  $adec(aenc(M, pk(A)), sk(A)) =_E M,$
  $check(sign(M, Sk), pk(Sk)) =_E ok$
  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 distinghis 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

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:

$$A \to B : \{N_A, A\}_{pk(B)} \quad (1)$$
$$B \to A : \{N_A, N_B\}_{pk(A)} \quad (2)$$
$$A \to B : \{N_B\}_{pk(B)} \quad (3)$$

[AIF-ω model]

Needham–Schröder, our Drosophila

Protocol:

$$A \to B : \{N_A, A\}_{pk(B)} \quad (1)$$
$$B \to A : \{N_A, N_B\}_{pk(A)} \quad (2)$$
$$A \to B : \{N_B\}_{pk(B)} \quad (3)$$

Deduction rules:

$$\frac{}{aenc(pair(N_A, A), pk(B))} \; (1)$$

$$\frac{aenc(pair(N_A, A), pk(B))}{aenc(pair(N_A, N_B), pk(A))} \; (2)$$

$$\frac{aenc(pair(N_A, N_B), pk(B))}{aenc(N_B, pk(A))} \; (3)$$

[AIF-ω model]

Needham–Schröder–Lowe (fixed)

Protocol:

$$A \to B : \{N_A, A\}_{pk(B)} \quad (1)$$

$$B \to A : \{N_A, N_B\}_{pk(A)} \quad (2)$$

$$A \to B : \{N_B\}_{pk(B)} \quad (3)$$

Attack trace:

$$A \to I : \{N_A, A\}_{pk(I)}$$

$$I \to B : \{N_A, A\}_{pk(B)}$$

$$B \to I : \{N_A, N_B\}_{pk(A)}$$

$$I \to A : \{N_A, N_B\}_{pk(A)}$$

$$A \to I : \{N_B\}_{pk(I)}$$

$$I \to B : \{N_B\}_{pk(B)}$$

[AIF-ω model]

Needham–Schröder–Lowe (fixed)

Protocol:

$$A \to B : \{N_A, A\}_{pk(B)} \quad (1)$$

$$B \to A : \{N_A, N_B, B\}_{pk(A)} \quad (2)$$

$$A \to B : \{N_B\}_{pk(B)} \quad (3)$$

[AIF-ω 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)...

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 appraoches 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 appraoches 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 $ring(a)$ of current keys, registered at the key server s_1, either in $valid(s_1, a)$ or in $revoked(s_1, a)$.

Set-Membership Abstraction: Key Ideas

- **Organize data into countable families of sets (data-bases), indexed by agent names.**
  Example: user a has a set $\mathit{ring}(a)$ of current keys, registered at the key server s_1, either in $\mathit{valid}(s_1, a)$ or in $\mathit{revoked}(s_1, a)$.
- **Disjointness assumption (for a fixed representation).**
  All sets of the same family are pairwise disjoint.
  E.g. $\forall A, B \in \mathit{User} \,.\, A \neq B \implies \mathit{ring}(A) \cap \mathit{ring}(B) = \emptyset$.
  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 $ring(a)$ of current keys, registered at the key server s_1, either in $valid(s_1, a)$ or in $revoked(s_1, a)$.
- **Disjointness assumption (for a fixed representation).**
  All sets of the same family are pairwise disjoint.
  E.g. $\forall A, B \in User \,.\, A \neq B \implies ring(A) \cap ring(B) = \emptyset$.
  Violations of the assumption constitute an attack.
- **Annotate the current state of an object with a term.**
  Given the ordering $\langle ring, valid, revoked \rangle$, a key pk satisfying only $pk \in valid(s_1, a)$ is annotated by $\langle 0, valid(s_1, a), 0 \rangle$

Set-Membership Abstraction: Key Ideas

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

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

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

- **Term implications: representing set-membership changes**

  If $pk@\overbrace{\langle 0, valid(s_1, a), 0\rangle}^{s} \twoheadrightarrow pk@\overbrace{\langle 0, 0, revoked(s_1, a)\rangle}^{t}$,
  then for every context $C[\cdot]$ where $C[s]$ holds, also $C[t]$ holds.


NAL MASOWY
GDANSK
1925
ENGE CUP

Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

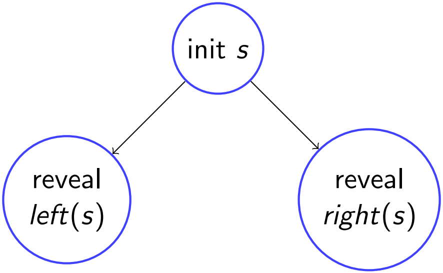
Examples

Translation into Horn Clauses

Conclusions

Hello World: a Hardware Security Module

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



[AIF-ω model]

AIF key revocation protocol [AIF-ω model]

Types:

$$\begin{aligned} Honest &= \{a, b, c\}; & Dishon &= \{i\}; \\ User &= Honest \cup Dishon; & Server &= \{s_1, s_2\}; \end{aligned}$$

Sets:

$$ring(User),\ \ valid(Server, User),\ \ revoked(Server, User);$$

AIF key revocation protocol [AIF-ω model]

Types:

$$\begin{array}{rclrcl} \mathit{Honest} & = & \{a,b,c\}; & \mathit{Dishon} & = & \{i\}; \\ \mathit{User} & = & \mathit{Honest} \cup \mathit{Dishon}; & \mathit{Server} & = & \{s_1, s_2\}; \end{array}$$

Sets:

$$\mathit{ring}(\mathit{User}),\ \mathit{valid}(\mathit{Server},\mathit{User}),\ \mathit{revoked}(\mathit{Server},\mathit{User});$$

Rules:

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

AIF key revocation protocol [AIF-ω model]

Types:

$$\begin{array}{rclcrcl} \mathit{Honest} & = & \{a, b, c\}; & & \mathit{Dishon} & = & \{i\}; \\ \mathit{User} & = & \mathit{Honest} \cup \mathit{Dishon}; & & \mathit{Server} & = & \{s_1, s_2\}; \end{array}$$

Sets:

$$\mathit{ring}(\mathit{User}),\ \mathit{valid}(\mathit{Server}, \mathit{User}),\ \mathit{revoked}(\mathit{Server}, \mathit{User});$$

Rules:

$$\begin{array}{l} \mathit{registerOutOfBand}(U : \mathit{User}, S : \mathit{Server}) = \\ \quad =\![PK]\!\Rightarrow PK \in \mathit{ring}(U) \cdot PK \in \mathit{valid}(S, U) \cdot \mathrm{i}(PK) \end{array}$$

$$\begin{array}{l} \mathit{updateKey}(U : \mathit{User}, S : \mathit{Server}, PK : \mathit{val}, NPK : \mathit{val}) = \\ \quad \mathrm{i}(\mathrm{sign}_{\mathrm{inv}(PK)}(S, U, NPK)) \cdot PK \in \mathit{valid}(S, U) \cdot \\ \quad NPK \notin \mathit{valid}(\_, \_) \cdot NPK \notin \mathit{revoked}(\_, \_) \\ \quad \Rightarrow PK \in \mathit{revoked}(S, U) \cdot NPK \in \mathit{valid}(S, U) \cdot \mathrm{i}(\mathrm{inv}(PK)) \end{array}$$

AIF key revocation protocol [AIF-ω model]

Types:

$$\begin{array}{rclcrcl} \mathit{Honest} & = & \{a, b, c\}; & & \mathit{Dishon} & = & \{i\}; \\ \mathit{User} & = & \mathit{Honest} \cup \mathit{Dishon}; & & \mathit{Server} & = & \{s_1, s_2\}; \end{array}$$

Sets:

$$\mathit{ring}(\mathit{User}),\ \mathit{valid}(\mathit{Server}, \mathit{User}),\ \mathit{revoked}(\mathit{Server}, \mathit{User});$$

Rules:

$\mathit{registerOutOfBand}(U : \mathit{User}, S : \mathit{Server}) =$
$\quad =\![PK]\!\Rightarrow PK \in \mathit{ring}(U) \cdot PK \in \mathit{valid}(S, U) \cdot \mathrm{i}(PK)$

$\mathit{updateKey}(U : \mathit{User}, S : \mathit{Server}, PK : \mathit{val}, NPK : \mathit{val}) =$
$\quad \mathrm{i}(\mathrm{sign}_{\mathrm{inv}(PK)}(S, U, NPK)) \cdot PK \in \mathit{valid}(S, U) \cdot$
$\quad NPK \notin \mathit{valid}(\_, \_) \cdot NPK \notin \mathit{revoked}(\_, \_)$
$\quad \Rightarrow PK \in \mathit{revoked}(S, U) \cdot NPK \in \mathit{valid}(S, U) \cdot \mathrm{i}(\mathrm{inv}(PK))$

$\mathit{attackDef}(U : \mathit{Honest}, S : \mathit{Server}, PK : \mathit{val}) =$
$\quad \mathrm{i}(\mathrm{inv}(PK)) \cdot PK \in \mathit{valid}(S, U) \Rightarrow \mathit{attack}.$

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

Types:

$$\begin{aligned} \mathit{Honest} &= \{a, b, c, \ldots\}; & \mathit{Dishon} &= \{i, \ldots\}; \\ \mathit{User} &= \mathit{Honest} \cup \mathit{Dishon}; & \mathit{Server} &= \{s_1, s_2, \ldots\}; \end{aligned}$$

Sets:

$$\mathit{ring}(\mathit{User}),\ \mathit{valid}(\mathit{Server}, \mathit{User}),\ \mathit{revoked}(\mathit{Server}, \mathit{User});$$

Rules:

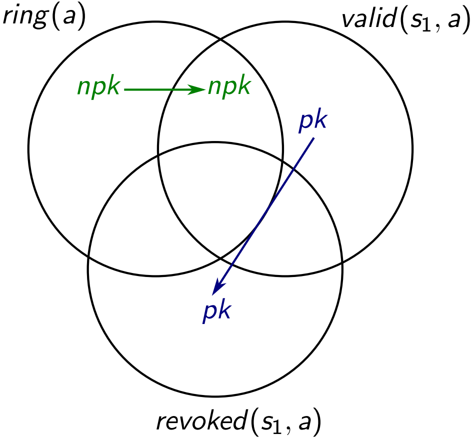
$\mathit{registerOutOfBand}(U : \mathit{User}, S : \mathit{Server}) =$
$\quad =[PK]\Rightarrow PK \in \mathit{ring}(U) \cdot PK \in \mathit{valid}(S, U) \cdot \mathrm{i}(PK)$

$\mathit{updateKey}(U : \mathit{User}, S : \mathit{Server}, PK : \mathit{val}, NPK : \mathit{val}) =$
$\quad \mathrm{i}(\mathrm{sign}_{\mathrm{inv}(PK)}(S, U, NPK)) \cdot PK \in \mathit{valid}(S, U) \cdot$
$\quad NPK \notin \mathit{valid}(\_, \_) \cdot NPK \notin \mathit{revoked}(\_, \_)$
$\quad \Rightarrow PK \in \mathit{revoked}(S, U) \cdot NPK \in \mathit{valid}(S, U) \cdot \mathrm{i}(\mathrm{inv}(PK))$

$\mathit{attackDef}(U : \mathit{Honest}, S : \mathit{Server}, PK : \mathit{val}) =$
$\quad \mathrm{i}(\mathrm{inv}(PK)) \cdot PK \in \mathit{valid}(S, U) \Rightarrow \mathit{attack}.$

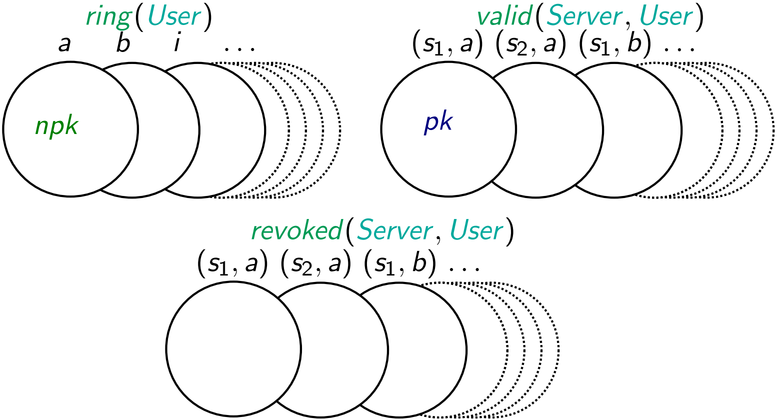
$updateKey(U : User, S : Server, PK : val, NPK : val) =$
$[\ldots] \cdot PK \in valid(S, U) \cdot NPK \notin valid(\_, \_) \cdot NPK \notin revoked(\_, \_)$
$\Rightarrow PK \in revoked(S, U) \cdot NPK \in valid(S, U) \cdot [\ldots]$

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



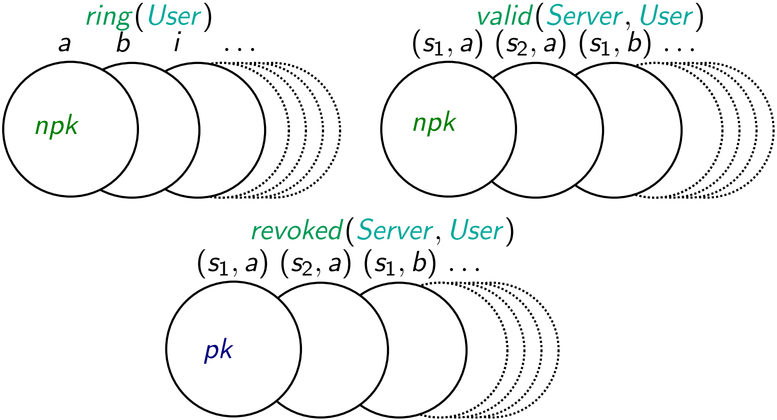
After: $pk@\langle 0, 0, revoked(s_1, a)\rangle$, $npk@\langle ring(a), valid(s_1, a), 0\rangle$

$updateKey(s_1, a)$



$$pk@\langle 0, valid(s_1, a), 0\rangle \twoheadrightarrow pk@\langle 0, 0, revoked(s_1, a)\rangle$$
$$npk@\langle ring(a), 0, 0\rangle \twoheadrightarrow npk@\langle ring(a), valid(s_1, a), 0\rangle$$

$updateKey(s_1, a)$



$$pk@\langle 0, valid(s_1, a), 0\rangle \twoheadrightarrow pk@\langle 0, 0, revoked(s_1, a)\rangle$$

$$npk@\langle ring(a), 0, 0\rangle \twoheadrightarrow npk@\langle ring(a), valid(s_1, a), 0\rangle$$

Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions

Inadmissible (stupid) rules

- Rules violating *disjointness* on RHS:

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

Inadmissible (stupid) rules

- Rules violating *disjointness* on RHS:

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

Can be compiled into:

$$\begin{aligned} r_1(X\colon val) = p(X) \cdot X \notin s_1(\_) &\implies X \in s_1(a) \\ r_2(X\colon val, A\colon T) = p(X) \cdot X \in s_1(A) &\implies \color{red}{attack} \end{aligned}$$

Inadmissible (stupid) rules

- Rules violating *disjointness* on RHS:

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

Can be compiled into:

$$\begin{array}{c} r_1(X\colon val) = p(X) \cdot X \notin s_1(\_) \implies X \in s_1(a) \\ r_2(X\colon val, A\colon T) = p(X) \cdot X \in s_1(A) \implies \textit{attack} \end{array}$$

- 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 $updateKey(U, S)$ into Horn clauses

1. Compute equivalence class

			EquivalenceClass	
$\mathrm{i}(\mathrm{sign}_{\mathrm{inv}(PK)}(S, U, NPK)) \cdot$				
$PK \in valid(S, U) \cdot$	PK	X_1	$valid(S, U)$	X_2
$NPK \notin valid(\_, \_) \cdot NPK \notin revoked(\_, \_)$	NPK	X_3	0	0
$\Rightarrow$				
$PK \in revoked(S, U) \cdot$	PK	X_1	0	$revoked(S, U)$
$NPK \in valid(S, U) \cdot$	NPK	X_3	$valid(S, U)$	0
$\mathrm{i}(\mathrm{inv}(PK))$				

Translating *updateKey*(U, S) into Horn clauses

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

$$\begin{array}{l}
\mathrm{i}(\mathrm{sign}_{\mathrm{inv}(\langle X_1, valid(S,U), X_2\rangle)}(S,U,\langle X_3, 0, 0\rangle)) \cdot \\
PK \in valid(S,U) \cdot \\
NPK \notin valid(_,_) \cdot NPK \notin revoked(_,_) \\
\Rightarrow \\
PK \in revoked(S,U) \cdot \\
NPK \in valid(S,U) \cdot \\
\mathrm{i}(\mathrm{inv}(\langle X_1, 0, revoked(S,U)\rangle))
\end{array}$$

		EquivalenceClass	
PK	X_1	$valid(S,U)$	X_2
NPK	X_3	0	0
PK	X_1	0	$revoked(S,U)$
NPK	X_3	$valid(S,U)$	0

Translating $updateKey(U, S)$ into Horn clauses

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

			EquivalenceClass	
$\mathrm{i}(\mathrm{sign}_{\mathrm{inv}(\langle X_1, valid(S,U), X_2\rangle)}(S, U, \langle X_3, 0, 0\rangle)) \cdot$				
$PK \in valid(S, U) \cdot$	PK	X_1	$valid(S, U)$	X_2
$NPK \notin valid(\_, \_) \cdot NPK \notin revoked(\_, \_)$	NPK	X_3	0	0
$\Rightarrow$				
$PK \in revoked(S, U) \cdot$	PK	X_1	0	$revoked(S, U)$
$NPK \in valid(S, U) \cdot$	NPK	X_3	$valid(S, U)$	0
$\mathrm{i}(\mathrm{inv}(\langle X_1, 0, revoked(S, U)\rangle))$				
$\langle X_1, valid(S, U), 0\rangle \twoheadrightarrow \langle X_1, 0, revoked(S, U)\rangle$				
$\langle X_3, 0, 0\rangle \twoheadrightarrow \langle X_3, valid(S, U), 0\rangle$				

Translating $updateKey(U, S)$ into Horn clauses

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

			EquivalenceClass	
$\mathrm{i}(\mathrm{sign}_{\mathrm{inv}(\langle X_1, valid(S,U), X_2\rangle)}(S, U, \langle X_3, 0, 0\rangle)) \cdot$				
$PK \in valid(S, U) \cdot$	PK	X_1	$valid(S, U)$	X_2
$NPK \notin valid(\_, \_) \cdot NPK \notin revoked(\_, \_)$	NPK	X_3	0	0
$\Rightarrow$				
$PK \in revoked(S, U) \cdot$	PK	X_1	0	$revoked(S, U)$
$NPK \in valid(S, U) \cdot$	NPK	X_3	$valid(S, U)$	0

$\mathrm{i}(\mathrm{inv}(\langle X_1, 0, revoked(S, U)\rangle))$

$\langle X_1, valid(S, U), 0\rangle \twoheadrightarrow \langle X_1, 0, revoked(S, U)\rangle$

$\langle X_3, 0, 0\rangle \twoheadrightarrow \langle X_3, valid(S, U), 0\rangle$

$$\forall C[\cdot] \,.\, \langle X_3, 0, 0\rangle \twoheadrightarrow \langle X_3, valid(S, U), 0\rangle \cdot C[\langle X_3, 0, 0\rangle] \implies C[\langle X_3, valid(S, U), 0\rangle]$$

Translating $\mathit{updateKey}(U, S)$ 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

$\mathit{usr}(U) \cdot \mathit{srv}(S)$
$\mathrm{i}(\mathrm{sign}_{\mathrm{inv}(\langle X_1, \mathit{valid}(S,U), X_2\rangle)}(S, U, \langle X_3, 0, 0\rangle)) \cdot$
$PK \in \mathit{valid}(S, U) \cdot$
$NPK \notin \mathit{valid}(\_, \_) \cdot NPK \notin \mathit{revoked}(\_, \_)$
$\Rightarrow$
$PK \in \mathit{revoked}(S, U) \cdot$
$NPK \in \mathit{valid}(S, U) \cdot$
$\mathrm{i}(\mathrm{inv}(\langle X_1, 0, \mathit{revoked}(S, U)\rangle))$
$\langle X_1, \mathit{valid}(S, U), 0\rangle \twoheadrightarrow \langle X_1, 0, \mathit{revoked}(S, U)\rangle$
$\langle X_3, 0, 0\rangle \twoheadrightarrow \langle X_3, \mathit{valid}(S, U), 0\rangle$

		EquivalenceClass	
PK	X_1	$\mathit{valid}(S, U)$	X_2
NPK	X_3	0	0
PK	X_1	0	$\mathit{revoked}(S, U)$
NPK	X_3	$\mathit{valid}(S, U)$	0

$$\forall C[\cdot] \,.\, \langle X_3, 0, 0\rangle \twoheadrightarrow \langle X_3, \mathit{valid}(S, U), 0\rangle \cdot C[\langle X_3, 0, 0\rangle] \implies C[\langle X_3, \mathit{valid}(S, U), 0\rangle]$$

Outline

Verifying Security Protocols

Needham Schröder

Stateful Verification

Examples

Translation into Horn Clauses

Conclusions

Experimental Results: Key Server Example

	Number of Agents			Backend	
	Honest	Dishon	Server	ProVerif	SPASS
AIF	1	1	1	0.025s	0.891s
	2	1	1	0.135s	324.696s
	2	2	1	0.418s	Timeout
	3	3	1	2.057s	Timeout
AIF-ω	ω	ω	ω	0.034s	0.941s

Conclusions

We extend a successful method in a successful way:

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

Thank you :)