Symbolic verification of cryptographic protocols using Tamarin

Part 3 : Security Properties and Algorithmic Verification

David Basin
ETH Zurich

Summer School on Verification Technology, Systems & Applications
Nancy France
August 2018
1 Protocol Security Goals

2 Automated Verification

3 Tamarin workflow
1 Protocol Security Goals

2 Automated Verification

3 Tamarin workflow
Goals
What the protocol should achieve, e.g.,

- **Authenticate** messages, binding them to their originator
- Ensure **timeliness** of messages (recent, fresh, ...)
- Guarantee **secrecy** of certain items (e.g., generated keys)
Protocol Goals

Goals
What the protocol should achieve, e.g.,

- **Authenticate** messages, binding them to their originator
- Ensure **timeliness** of messages (recent, fresh, ...)
- Guarantee **secrecy** of certain items (e.g., generated keys)

Most common goals

- secrecy
- authentication (many different forms)

Other goals

- anonymity, non-repudiation (of receipt, submission, delivery),
  fairness, availability, sender invariance, ...
Protocol Properties and Correctness

What does it mean?

Properties

- Semantics of protocol $P$ is a set of traces: $\|P\| = \text{traces}(P)$.
  (Traces may be finite or infinite, state- or event-based.)
- Security goal / property $\phi$ also denotes a set of traces $\|\phi\|$.

Correctness has an exact meaning

- Protocol $P$ satisfies property $\phi$, written $P \models \phi$, iff $\|P\| \subseteq \|\phi\|$.
- Attack traces are those in $\|P\| - \|\phi\|$.
- Every correctness statement is true or false.

Later: how to determine which holds

Ok, no attacks.

Ok.
Protocol Properties and Correctness

What does it mean?

Properties

- Semantics of protocol $P$ is a set of traces: $\|P\| = \text{traces}(P)$. (Traces may be finite or infinite, state- or event-based.)
- Security goal / property $\phi$ also denotes a set of traces $\|\phi\|$.

Correctness has an exact meaning

- Protocol $P$ satisfies property $\phi$, written $P \models \phi$, iff $\|P\| \subseteq \|\phi\|$.
- Attack traces are those in $\|P\| - \|\phi\|$
- Every correctness statement is true or false. Later: how to determine which holds

Ok, no attacks.
Property specification language: a fragment of a many-sorted first-order logic with a sort for timepoints (prefixed with #), where quantification is over both messages and timepoints:

- **All** for universal quantification (temporal variables are prefixed with #)
- **Ex** for existential quantification (temporal variables are prefixed with #)
- **===>** for implication, **not** for negation
- **|** for disjunction ("or"), **&** for conjunction ("and")
- **f @ i** for action constraints (the sort prefix for the temporal variable 'i' is optional)
- **i < j** for temporal ordering (the sort prefix for the temporal variables 'i' and 'j' is optional)
- **#i = #j** for an equality between temporal variables 'i' and 'j'
- **x = y** for an equality between message variables 'x' and 'y'
The property that the fresh value $\sim n$ is distinct in all applications of a fictitious rule:

```
lemma distinct_nonces:
    "All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```

or equivalently

```
lemma distinct_nonces:
    all-traces
    "All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```

These lemmas require that the property holds for all traces, we can also express that there exists a trace for which the property holds:

```
lemma distinct_nonces:
    exists-trace
    "not All n #i #j. Act1(n)@i & Act1(n)@j ==> #i=#j"
```
• All action fact symbols may be used in formulas.
• All variables must be guarded.

**Guardedness**

For universally quantified variables:
• all variables must occur in an action constraint right after the quantifier and
• the outermost logical operator inside the quantifier must be an implication

For existentially quantified variables:
• all variables must occur in an action constraint right after the quantifier and
• the outermost logical operator inside the quantifier must be a conjunction
Direct formulation

- Formulate property $\phi$ directly in terms of actions occurring in protocol traces, i.e., as a set of (or predicate on) traces.
- Drawback: standard properties like secrecy and authentication become highly protocol-dependent, since they need to refer to the concrete protocol messages.
Formalizing Security Properties

Two approaches

Direct formulation

- Formulate property $\phi$ directly in terms of actions occurring in protocol traces, i.e., as a set of (or predicate on) traces.
- Drawback: standard properties like secrecy and authentication become highly protocol-dependent, since they need to refer to the concrete protocol messages.

Protocol instrumentation

- Idea: insert special claim events into the protocol roles:
  \[
  \text{Claim\_claimtype}(R, t),
  \]
  where $R$ is the executing role, claimtype indicates the type of claim, and $t$ is a message term.
- Interface for expressing properties independently of protocol.
- Example: Claim\_secret($A$, $N_A$) claims that $N_A$ is a secret for role $A$, i.e., not known to the intruder.
Claim events are part of the protocol rules as actions.

Properties of claim events

- Their only effect is to record facts (claims) in protocol trace.
- Intruder cannot have observed, modified, or generated them.
Claim events are part of the protocol rules as actions.

**Properties of claim events**

- Their only effect is to record facts (claims) in protocol trace.
- Intruder cannot have observed, modified, or generated them.

**Expressing properties using claim events**

- Properties of traces \( tr \) expressed in terms of claim events and other actions (e.g., adversary knowledge K) occurring in \( tr \).
- Properties are formulated from the point of view of a given role, thus yielding security guarantees for that role.
- We concentrate on secrecy and variants of authentication, although the approach is not limited to these properties.
Definition (Secrecy, informally)
The intruder cannot discover data (e.g., key, nonce, etc.) that is intended to be secret.

Role instrumentation
- Insert the claim event $\text{Claim}_{\text{secret}}(A, M)$ into role $A$ to claim that the message $M$ used in the run remains secret.
- Position: at the end of the role.
- For instance, in NSPK, the nonces $na$ and $nb$ should remain secret.

Note: In graphs, where the executing role is clear from context, we abbreviate $\text{Claim}_{\text{claimtype}}(A, t)$ to $\text{claimtype}(t)$ inside a hexagon.
Definition (Secrecy, first attempt)
The secrecy property consists of all traces \( tr \) satisfying

\[
\forall A, M, i. \text{Claim}_{-}\text{secret}(A, M)@i \Rightarrow \neg(\exists j. K(M)@j)
\]

- Let \( tr = tr_1; tr_2; \ldots; tr_k \) and \( x@k \) is shorthand for \( x \in tr_k \).
Definition (Secrecy, first attempt)

The secrecy property consists of all traces $tr$ satisfying

$$\forall A, M, i. \text{Claim}_\text{secret}(A, M)@i \Rightarrow \neg(\exists j. K(M)@j)$$

- Let $tr = tr_1; tr_2; \ldots; tr_k$ and $x@k$ is shorthand for $x \in tr_k$.
- Can only require $M$ to remain secret if $A$ runs the protocol with another honest agent, i.e.,
- Trivially broken if $A$ or $B$ is instantiated with a compromised agent, since then the adversary rightfully knows $M$.
- This definition is fine for a passive adversary, who observes network traffic, but does not participate in the protocol.
**Definition (Compromised Agent)**

A *compromised agent* is under adversary control. It shares all its information with the adversary and can participate in protocols. We model this by having the agent give its initial secret information to the adversary, which can then mimic the agent’s actions.

We note the fact that an agent is compromised by a *Rev* event in the trace, attached to the rule that reveals its initial secrets to the adversary (compare to the creation rule):

\[ !\text{Ltk}(A, skA) \xrightarrow{\text{Rev}(A)} \text{Out}(skA) \]

**Exercise:** convince yourself that, given the agent’s secret, the adversary can perform all of the agent’s send and receive steps.
Definition (Honesty)

An agent $A$ is **honest** in a trace $tr$ when $\text{Rev}(A) \notin tr$. When making a claim in a rule action, all parties $B$ that are expected to be honest must be listed with a $\text{Honest}(B)$ action in that rule.
Definition (Honesty)

An agent $A$ is honest in a trace $tr$ when $\text{Rev}(A) \not\in tr$. When making a claim in a rule action, all parties $B$ that are expected to be honest must be listed with a $\text{Honest}(B)$ action in that rule.

Definition (Secrecy)

The secrecy property consists of all traces $tr$ satisfying

$$\forall A \ M \ i. \ (\text{Claim}_\text{secret}(A, M)@i)$$

$$\Rightarrow (\neg(\exists j. \ K(M)@j) \lor (\exists B \ j. \ \text{Rev}(B)@j \land \text{Honest}(B)@i))$$
Secrecy Example #1

**msc** Secrecy for Symmetric Encryption

- This is fine: secrecy holds for both A and B.
- We omit the obvious annotations Honest(A), Honest(B) in message sequence charts for 2-party protocols.
Secrecy Example #1

This is fine: secrecy holds for both $A$ and $B$.
This is fine: secrecy holds for both $A$ and $B$.

We omit the obvious annotations Honest($A$), Honest($B$) in message sequence charts for 2-party protocols.
Secrecy Example #2

**msc** Secrecy for Asymmetric Encryption

- **A**
  -密文: \(\{A, N_A\}_{pk(B)}\)
- **B**
  -秘密: \(\text{secret}(N_A)\)

- Secrecy holds for A: she knows that only B can decrypt message.
Secrecy Example #2

- Secrecy holds for A: she knows that only B can decrypt message.
**Secrecy Example #2**

---

**msc** Secrecy for Asymmetric Encryption

- Secrecy fails for \( B \): he does not know who encrypted message!
- N.B. Intruder can build and send message (without reveal).

---
Which authentication are you talking about?

- No unique definition of authentication, but a variety of different forms.
- Considerable effort spent on specifying and classifying, semi-formally or formally, different forms of authentication (e.g., by Cervesato/Syverson, Clark/Jacob, Gollmann, Lowe, Cremers et al.).

Examples

- ping authentication, aliveness, weak agreement, non-injective agreement, injective agreement, weak and strong authentication, synchronization, and matching histories.
\textbf{msc} Needham-Schroeder protocol

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

\[\{N_A, N_B\}_{pk(A)}\]

\[\{N_B\}_{pk(B)}\]
A More Realistic Picture
Successful Authentication
Gavin Lowe has defined the following hierarchy of increasingly stronger authentication properties:\footnote{Terminology and notation slightly adapted to our setting. Note that if either $a$ or $b$ is not honest, then the properties are said to trivially hold.}

**Aliveness** A protocol guarantees to an agent $a$ in role $A$ aliveness of another agent $b$ if, whenever $a$ completes a run of the protocol, apparently with $b$ in role $B$, then $b$ has previously been running the protocol.

**Weak agreement** A protocol guarantees to an agent $a$ in role $A$ weak agreement with another agent $b$ if, whenever agent $a$ completes a run of the protocol, apparently with $b$ in role $B$, then $b$ has previously been running the protocol, apparently with $a$. 
Non-injective agreement A protocol guarantees to an agent $a$ in role $A$ non-injective agreement with an agent $b$ in role $B$ on a message $M$ if, whenever $a$ completes a run of the protocol, apparently with $b$ in role $B$, then $b$ has previously been running the protocol, apparently with $a$, and $b$ was acting in role $B$ in his run, and the two principals agreed on the message $M$.

Injective agreement is non-injective agreement where additionally each run of agent $a$ in role $A$ corresponds to a unique run of agent $b$.

Variants may include recentness: insist that $B$’s run was, e.g., within $t$ time units.
**Non-injective agreement** A protocol guarantees to an agent $a$ in role $A$ non-injective agreement with an agent $b$ in role $B$ on a message $M$ if, whenever $a$ completes a run of the protocol, apparently with $b$ in role $B$, then $b$ has previously been running the protocol, apparently with $a$, and $b$ was acting in role $B$ in his run, and the two principals agreed on the message $M$.

**Injective agreement** is non-injective agreement where additionally each run of agent $a$ in role $A$ corresponds to a unique run of agent $b$.

Variants may include recentness: insist that $B$’s run was, e.g., within $t$ time units.

**How can we formalize these nontrivial properties?**
We use two claims to express that role $A$ authenticates role $B$ on $t$:

**In role $A$:**

- Insert a **commit claim event** $\text{Claim\_commit}(A, B, t), \text{Honest}(A), \text{Honest}(B)$.
- Position: after $A$ can construct $t$. Typically, at end of $A$’s role.

**In role $B$:**

- Insert a **running claim event** $\text{Claim\_running}(B, A, u)$.
- Term $u$ is $B$’s view of $t$.
- Position: after $B$ can construct $u$ and preceding $\text{Claim\_commit}(A, B, t)$. 
Formalizing Authentication

Definition (Non-injective agreement)

The property $\text{Agreement}_{NI}(A, B, t)$ consists of all traces satisfying

$$\forall a \ b \ t \ i. \ \text{Claim\_commit}(a, b, \langle A, B, t \rangle)@i$$
$$\Rightarrow (\exists j. \text{Claim\_running}(b, a, \langle A, B, t \rangle)@j)$$
$$\lor (\exists X \ r. \text{Rev}(X)@r \land \text{Honest}(X)@i)$$

- Whenever a commit claim is made with honest agents $a$ and $b$, then the peer $b$ must be running with the same parameter $t$, or the adversary has compromised at least one of the two agents.
Definition (Non-injective agreement)

The property $\text{Agreement}_{NI}(A, B, t)$ consists of all traces satisfying

$$\forall a \ b \ t \ i. \ \text{Claim\_commit}(a, b, \langle A, B, t \rangle)@i$$
$$\Rightarrow (\exists j. \text{Claim\_running}(b, a, \langle A, B, t \rangle)@j)$$
$$\lor (\exists X \ r. \text{Rev}(X)@r \land \text{Honest}(X)@i)$$

- Whenever a commit claim is made with honest agents $a$ and $b$, then the peer $b$ must be running with the same parameter $t$, or the adversary has compromised at least one of the two agents.

Exercise  Does ordering of $i$ and $j$ matter.
(Hint: set of traces is prefix closed.)
**Example: NSL Protocol (1/2)**

**msc NSL, instrumented for A to agree with B on $N_A, N_B$**

```
msc NSL, instrumented for A to agree with B on NA, NB
```

- **A**
  - $\{N_A, A\}_{pk(B)}$
  - $\text{running}(A, N_A, N_B)$
  - $\{N_A, N_B, B\}_{pk(A)}$
  - $\text{commit}(B, N_A, N_B)$
  - $\{N_B\}_{pk(B)}$

- **B**
  - $\{N_A, A\}_{pk(B)}$

```
A B
{NA, A}pk(B)
running(A, NA, NB)
{NA, NB, B}pk(A)
commit(B, NA, NB)
{NB}pk(B)
msc NSL, instrumented for A to agree with B on NA, NB
```
Example: NSL Protocol (2/2)

\[
\text{msc NSL, instrumented for } B \text{ to agree with } A \text{ on } N_A, N_B
\]

\[
\text{A} \quad \text{B}
\]

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

\[
\{N_A, N_B\}_{pk(A)}
\]

\[
\text{running}(B, N_A, N_B)
\]

\[
\{N_B\}_{pk(B)}
\]

\[
\text{commit}(A, N_A, N_B)
\]
Event causality in multi-hop authentication claims: The *running* event must causally precede the *commit* event and the messages $t$ and $u$ must be known at the position of the claim event in the respective role.
Example: Yahalom Protocol (1/3)

**msc Yahalom protocol**

**Initiator**
- A

**Responder**
- B

**Server**
- S

- \( A, N_A \)
- \( B, \{ A, N_A, N_B \}_{k(B,S)} \)
- \( \{ B, K_{AB}, N_A, N_B \}_{k(A,S)}, \{ A, K_{AB} \}_{k(B,S)} \)
- \( \{ A, K_{AB} \}_{k(B,S)}, \{ N_B \}_{K_{AB}} \)
**Example: Yahalom Protocol (2/3)**

```plaintext
msc Yahalom protocol (instrumented for responder authenticating initiator on \(N_A, N_B, K_{AB}\))
```

![Diagram of Yahalom protocol](image)

- **Initiator** (A)
- **Responder** (B)
- **Server** (S)

1. A, \(N_A\)
2. B, \(\{A, N_A, N_B\}_{k(B,S)}\)
3. \(\{B, K_{AB}, N_A, N_B\}_{k(A,S)}, \{A, K_{AB}\}_{k(B,S)}\)
4. \(\{A, K_{AB}\}_{k(B,S)}, \{N_B\}_{K_{AB}}\)
5. \(commit(A, N_A, N_B, K_{AB})\)

---

**Note:** The diagram illustrates the interaction between the Initiator, Responder, and Server in the Yahalom protocol, showing the flow of messages and keys used for authentication and commitment.

---

**msc** Yahalom protocol (instrumented for responder authenticating initiator on \(N_A, N_B, K_{AB}\))

- **Initiator** (A)
- **Responder** (B)
- **Server** (S)

1. A, \(N_A\)
2. B, \(\{A, N_A, N_B\}_{k(B,S)}\)
3. \(\{B, K_{AB}, N_A, N_B\}_{k(A,S)}, \{A, K_{AB}\}_{k(B,S)}\)
4. \(\{A, K_{AB}\}_{k(B,S)}, \{N_B\}_{K_{AB}}\)
5. \(commit(A, N_A, N_B, K_{AB})\)
**Example: Yahalom Protocol (3/3)**

**msc** Yahalom protocol (instrumented for initiator authenticating responder on $N_A, N_B$)

**Note:** agreement for $A$ on $K_{AB}$ is not possible, since $B$ gets $K_{AB}$ after $A$. 

\[
A, N_A \quad \text{running}(A, N_A, N_B) \quad B, \{A, N_A, N_B\}_{k(B,S)} \\
\{B, K_{AB}, N_A, N_B\}_{k(A,S)}, \{A, K_{AB}\}_{k(B,S)} \quad \textbf{commit}(B, N_A, N_B) \\
\{A, K_{AB}\}_{k(B,S)}, \{N_B\}_{K_{AB}} \quad B, \{A, N_A, N_B\}_{k(B,S)} \\
\]
Definition (Injective agreement)

The property $\text{Agreement}(A, B, t)$ consists of all traces satisfying:

\[
\forall a \ b \ t \ i. \ \text{Claim-commit}(a, b, \langle A, B, t \rangle)@i \\
\Rightarrow (\exists j. \text{Claim-running}(b, a, \langle A, B, t \rangle)@j \land j < i \\
\land \neg (\exists a_2 b_2 i_2. \text{Claim-commit}(a_2, b_2, \langle A, B, t \rangle)@i_2 \\
\land \neg (i_2 = i))) \\
\lor (\exists X \ r. \text{Rev}(X)@r \land \text{Honest}(X)@i)
\]

Remarks

- For each commit by $a$ in role $A$ on the trace there is a unique matching $b$ executing role $B$. 
Failed Injective Authentication
Successful Injective Authentication
Injective vs Non-injective Agreement

They are not equivalent

- Non-injective agreement holds.
- Injective agreement fails, since the adversary can replay message to several threads in responder role B.
Injective Agreement counter-example
Definition (Weak agreement)

A trace $tr$ satisfies the property $WeakAgreement(A, B)$ iff

$$\forall a\ b\ i. \ Claim\_commit(a, b, \langle \rangle)@i \Rightarrow (\exists j. Claim\_running(b, a, \langle \rangle)@j)$$

$$\lor (\exists X \ r. Rev(X)@r \land Honest(X)@i)$$

It is sufficient that the agents agree they are communicating, it is not required that they play the right roles. Note also the empty list of data $\langle \rangle$ that is agreed upon, i.e., none.
Definition (Aliveness)

A trace $tr$ satisfies the property $Alive(A, B)$ iff

$$\forall a \ b \ i. \ \text{Claim-commit}(a, b, \langle \rangle)@i \Rightarrow (\exists j \ \text{id}. \text{Create}_B(b, \text{id})@j \lor \text{Create}_A(b, \text{id})@j) \lor (\exists X \ r. \text{Rev}(X)@r \land \text{Honest}(X)@i)$$

It is neither required that the agent $b$, believed to instantiate role $B$ by agent $a$, really plays role $B$, nor that he believes to be talking to $a$. 
Aliveness vs Weak Agreement

They are not equivalent

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

\[ \text{running}(A) \]

\[ N_A \]

\[ \text{commit}(B) \]
Aliveness vs Weak Agreement

They are not equivalent

- **Aliveness holds**: only $B$ can decrypt the fresh nonce $N_A$.
- **Weak agreement fails**, since adversary may modify unprotected identity $A$ to $C$ in first message so that $B$ thinks he is talking to $C$. 
Weak Agreement counter-example
**msc** Mutual authentication protocol

- **Initiator** (A)
  - \(\{|N_A\}_k(A, B)\)
  - \(k(A, B)\)

- **Responder** (B)
  - \(\{|N_B\}_k(A, B), N_A\)
  - \(\text{running}(A)\)
  - \(\text{commit}(B)\)
  - \(N_B\)

**Reflection attack:** A may complete run without B’s participation.

Hence, aliveness fails.
When Even Aliveness Fails ...

MSC Mutual authentication protocol

- **Initiator**
  - A
  - $\{N_A\}_{k(A,B)}$

- **Responder**
  - B
  - $\{N_B\}_{k(A,B)}, N_A$
  - commit(B)
  - $N_B$

- Reflection attack: A may complete run without B’s participation.
- Hence, aliveness fails.
Attack found by Tamarin
Basic key-oriented goals

- key freshness

- (implicit) key authentication: a key is only known to the communicating agents $A$ and $B$ and mutually trusted parties

- key confirmation of $A$ to $B$ is provided if $B$ has assurance that agent $A$ has possession of key $K$

- explicit key authentication $=$ key authentication $+$ key confirmation $\Rightarrow$ expressible in terms of secrecy and agreement
Basic key-oriented goals

- key freshness
- (implicit) key authentication: a key is only known to the communicating agents $A$ and $B$ and mutually trusted parties
- key confirmation of $A$ to $B$ is provided if $B$ has assurance that agent $A$ has possession of key $K$
- explicit key authentication = key authentication + key confirmation $\Rightarrow$ expressible in terms of secrecy and agreement

Goals concerning compromised keys

- (perfect) forward secrecy: compromise of long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals
- resistance to key-compromise impersonation: compromise of long-term key of an agent $A$ does not allow the adversary to masquerade to $A$ as a different principal.
• Signatures are used to authenticate the Diffie-Hellman public keys $\exp(g, X)$ and $\exp(g, Y)$.
• Signatures are used to authenticate the Diffie-Hellman public keys $\exp(g, X)$ and $\exp(g, Y)$.
• Protocol provides forward secrecy: The adversary cannot derive session key $K_{AB} = \exp(\exp(g, X), Y)$ by compromise of signing keys.
• Message exchange as in basic DH; protocol combines long-term and ephemeral DH keys to authenticate exchanged DH public keys.
- Message exchange as in basic DH; protocol combines long-term and ephemeral DH keys to authenticate exchanged DH public keys.
- Protocol does not provide forward secrecy: The adversary can construct the session key $K_{AB} = g^{VX + UY}$ as $(g^X)^V \cdot (g^Y)^U$ from observed messages and long-term private keys $U$ and $V$. 

\[ K_{AB} = (g^Y)^U \cdot (g^V)^X \]

\[ K_{AB} = (g^X)^V \cdot (g^U)^Y \]
• A generates an ephemeral asymmetric key pair \((\text{pk}(K_T), \text{sk}(K_T))\).
A generates an ephemeral asymmetric key pair \((\text{pk}(K_T), \text{sk}(K_T))\).

Protocol provides **forward secrecy** without using **Diffie-Hellman** keys: Adversary cannot learn session key by compromise of signing keys.

**MSC** Key transport protocol providing forward secrecy

\[
\begin{align*}
\text{A} & \quad (\text{pk}(K_T), \text{sk}(K_T)) \\
& \quad A, N_A, \{\text{pk}(K_T), B\}_{\text{sk}(A)} \\
& \quad \{K_{AB}\}_{\text{pk}(K_T)}, \{h(K_{AB}), A, N_A\}_{\text{sk}(B)} \\
\text{B} & \quad K_{AB}
\end{align*}
\]
1 Protocol Security Goals

2 Automated Verification

3 Tamarin workflow
We would like to have a program $V$ with . . .

- **Input:**
  - some description of a program $P$
  - some description of a functional specification $S$

- **Output:** *Yes* if $P$ satisfies $S$, and *No* otherwise.

- **Optional extra:** in the *No* case, give a counter-example, i.e. an input on which $P$ violates the specification.
We would like to have a program $V$ with . . .

- **Input:**
  - some description of a program $P$
  - some description of a functional specification $S$
- **Output:** *Yes* if $P$ satisfies $S$, and *No* otherwise.
- **Optional extra:** in the *No* case, give a counter-example, i.e. an input on which $P$ violates the specification.

Forget it:

**Theorem (Rice)**

*Let $S$ be any non-empty, proper subset of the computable functions. Then the verification problem for $S$ (the set of programs $P$ that compute a function in $S$) is undecidable.*
For security protocols, the \textit{state space} can be infinite for (at least) the following reasons:

\begin{itemize}
  \item \textbf{Messages} \quad The intruder can compose arbitrarily complex messages from his knowledge, e.g., \(i, h(i), h(h(i)), \ldots\).
  \item \textbf{Sessions} \quad Any number of sessions (or threads) may be executed.
  \item \textbf{Nonces} \quad Unbounded number of fresh nonces generated.
\end{itemize}
(Un)decidability: Complete picture

Bottom line: need at least two bounded parameters for decidability.
1. Protocol Security Goals
2. Automated Verification
3. Tamarin workflow
• Use multiset rewriting to represent protocol
• Adversary message deduction rules given as multiset rewriting rules
• Properties specified in first-order logic
  • Allows quantification over messages and timepoints
• Verification algorithm is proven sound and complete
• Use multiset rewriting to represent protocol
• Adversary message deduction rules given as multiset rewriting rules
• Properties specified in first-order logic
  • Allows quantification over messages and timepoints
• Verification algorithm is proven sound and complete

• Backwards reachability analysis – searching for insecure states
  • Negate security property, search for solutions
• Constraint solving
  • Uses dependency graphs
  • Normal dependency graphs for state-space reduction – efficiency despite undecidability
Dependency graph example

1: \( Fr(a) \rightarrow Fr(k) \)
2: \( Fr(a) \rightarrow Fr(k) \)
3: \( St(a, k) \rightarrow Out(enc(a, k)) \rightarrow Key(k) \)
4: \( Key(k) \rightarrow [Rev(k)] \)
5: \( Out(enc(a, k)) \rightarrow K(enc(a, k)) \)
6: \( Out(k) \rightarrow K(k) \)
7: \( K(enc(a, k)) \rightarrow K(k) \)
8: \( K(a) \rightarrow K(\langle a, a \rangle) \)
9: \( K(\langle a, a \rangle) \rightarrow [K(\langle a, a \rangle)] \)
10: \( St(a, k) \rightarrow In(\langle a, a \rangle) \rightarrow [Fin(a, k)] \)
Tamarin’s constraint solving algorithm

1: function \textsc{Solve}(P \models_{E_DH} \varphi)
2: \hspace{1em} \hat{\varphi} \leftarrow \neg \varphi \text{ rewritten into negation normal form}
3: \hspace{1em} \Omega \leftarrow \{\{\hat{\varphi}\}\}
4: \hspace{1em} \textbf{while } \Omega \neq \emptyset \text{ and solved}(\Omega) = \emptyset \textbf{ do}
5: \hspace{2em} \textbf{choose } \Gamma \sim_P \{\Gamma_1, \ldots, \Gamma_k\} \text{ such that } \Gamma \in \Omega
6: \hspace{2em} \Omega \leftarrow (\Omega \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}
7: \hspace{1em} \textbf{if } \text{ solved}(\Omega) \neq \emptyset
8: \hspace{2em} \textbf{then return} “attack(s) found: ”, solved(\Omega)
9: \hspace{2em} \textbf{else return} “verification successful”
Some constraint solving rules

Trace formula reduction rules:

\[ S_\approx : \quad \Gamma \rightsquigarrow_P \sigma, t, t_1, t_2 \text{Unify}_{AC} (\Gamma, \sigma) \]
\[ S_\bot : \quad \Gamma \rightsquigarrow_P \bot \]
\[ S_{\bot,=} : \quad \Gamma \rightsquigarrow_P \bot \]
\[ S_{\top,=} : \quad \Gamma \rightsquigarrow_P \top \]
\[ S_{\bot,\bot} : \quad \Gamma \rightsquigarrow_P \bot \]
\[ S_{\bot,a} : \quad \Gamma \rightsquigarrow_P (i < j, \Gamma) \]
\[ S_{\top} : \quad \Gamma \rightsquigarrow_P (\phi_1, \phi_2) \]
\[ S_{\neg} : \quad \Gamma \rightsquigarrow_P (\phi_1, \phi_2, \Gamma) \]
\[ S_{\exists} : \quad \Gamma \rightsquigarrow_P (\phi \{y/x\}, \Gamma) \quad \text{if } (\exists x:s. \phi) \in \Gamma, \phi \{w/x\} \notin_{AC} \Gamma \text{ for every term } w \text{ of sort } s, \text{ and } y:s \text{ fresh} \]
\[ S_{\forall} : \quad \Gamma \rightsquigarrow_P (\psi \sigma, \Gamma) \quad \text{if } (\forall x. -(f @i) \lor \psi) \in \Gamma, \text{ dom}(\sigma) = \text{set}(\bar{x}), (f @i) \sigma \notin_{AC} \Gamma \text{ as}(\Gamma), \text{ and } \psi \sigma \notin_{AC} \Gamma \]
Some more constraint solving rules

Graph constraint reduction rules:

\[ \text{U}_{bl} : \quad \Gamma \Rightarrow_P (r_i \approx r_i', \Gamma) \quad \text{if} \quad \{i : r_i, i : r_i'\} \subseteq \Gamma \quad \text{and} \quad r_i \neq AC r_i' \]

\[ \text{DG}_{1} : \quad \Gamma \Rightarrow_P \bot \quad \text{if} \quad i \ll_{\Gamma} i \]

\[ \text{DG}_{2} : \quad \Gamma \Rightarrow_P (f \approx f', \Gamma) \quad \text{if} \quad c \Rightarrow p \in \Gamma, (c, f) \in cs(\Gamma), (p, f') \in ps(\Gamma), \text{and} \quad f \neq AC f' \]

\[ \text{DG}_{3} : \quad \Gamma \Rightarrow_P \left( \text{if} \quad u = v \quad \text{then} \quad \Gamma \{i/j\} \quad \text{else} \quad \bot \right) \quad \text{if} \quad \{(i, v) \Rightarrow p, (j, u) \Rightarrow p\} \subseteq \Gamma \quad \text{and} \quad i \neq j \]

\[ \text{DG}_{4} : \quad \Gamma \Rightarrow_P \Gamma \{i/j\} \quad \text{if} \quad \{(i : r_i, (i, u) \Rightarrow p) \text{, } \Gamma \} \quad \text{if} \quad \{i : r_i, (i, u) \Rightarrow p, \Gamma\} \quad \text{if} \quad p \text{ is an open } f \text{-premise in } \Gamma, f \text{ is not a } K^l \text{- or } K^{\bot} \text{-fact, and } i \text{ fresh} \]

\[ \text{DG}_{3} : \quad \Gamma \Rightarrow_P \Gamma \{i/j\} \quad \text{if} \quad \{c \Rightarrow (i, v), c \Rightarrow (j, u)\} \subseteq \Gamma, c \text{ linear in } \Gamma, \text{ and } i \neq j, \]

\[ \text{DG}_{4} : \quad \Gamma \Rightarrow_P \Gamma \{i/j\} \quad \text{if} \quad \{i : \text{Fr}(m), j : \text{Fr}(m)\} \subseteq AC \Gamma \quad \text{and} \quad i \neq j \]

\[ \text{N}_{1} : \quad \Gamma \Rightarrow_P \bot \quad \text{if} \quad (i : r_i) \in \Gamma \quad \text{and} \quad r_i \text{ not } \Downarrow_{DH}-\text{normal} \]

\[ \text{N}_{5,6} : \quad \Gamma \Rightarrow_P \Gamma \{i/j\} \quad \text{if} \quad \{((i, 1), K^d_e(t)), ((j, 1), K^{d'}_e(t))\} \subseteq AC \quad CS(\Gamma), \quad i \neq j, \quad \text{and} \]

\[ d = d' \text{ or } \{i, j\} \cap \{k \mid \exists ri \in \text{insts}(\{\text{PAIR} \uparrow, \text{INV} \uparrow \text{, COERCENCE}\}) \quad \text{.} \quad (k : r_i) \in \Gamma = \emptyset \]

\[ \text{N}_{6} : \quad \Gamma \Rightarrow_P (i \ll_{\Gamma} j, \Gamma) \quad \text{if} \quad ((j, v), K^d_e(t)) \in ps(\Gamma), m \in AC \text{ inp}(t), ((i, u), K^e(t)) \in cs(\Gamma), \text{ and not } i \ll_{\Gamma} j \]

\[ \text{N}_{7} : \quad \Gamma \Rightarrow_P \bot \quad \text{if} \quad (i : K^d_e(s_1), K^t_e(s_1) \Rightarrow \text{Fr}(s_2 \cdot t_2)) \in \Gamma, \quad s_2 \text{ is of sort pub, and } \text{inp}(t_2) \subseteq \text{inp}(t_1) \]

Message deduction constraint reduction rules:

\[ \text{DG}_{2, t_{i}} : \quad \Gamma \Rightarrow_P (l \Rightarrow K^d_e(t)) \in ND \quad \text{if} \quad p \text{ is an open implicit } m \text{-construction in } \Gamma, m \text{ non-trivial, and } i \text{ fresh} \]

\[ \text{DG}_{2, t_{e}} : \quad \Gamma \Rightarrow_P (i : r_i, (i, 1) \Rightarrow p, \Gamma) \quad \text{if} \quad p \text{ is an open } K^e(m) \text{-premise in } \Gamma, \quad \{m\} = \text{inp}(m), m \text{ non-trivial, and } i \text{ fresh} \]

\[ \text{DG}_{2, t} : \quad \Gamma \Rightarrow_P (i : \text{Out}(y) \Rightarrow \text{Fr}(y), (i, 1) \Rightarrow p, \Gamma) \quad \text{if} \quad p \text{ is an open } K^e(m) \text{-premise in } \Gamma \text{ and } y, i \text{ fresh} \]

\[ \text{DG}_{2, \rightarrow} : \quad (c \Rightarrow p, \Gamma) \Rightarrow (c \Rightarrow p, \Gamma) \quad \text{if} \quad (c, K^e(m)) \in cs(\Gamma), m \notin \mathcal{V}_{msg}, \text{ and } i \text{ fresh} \]
Tamarin workflow

Equational theory $\mathcal{E}$:
builtins: bilinear-pairing, multiset
functions: h/1, pair/2, fst/1, snd/1
equations: \( \text{fst}(\text{pair}(x,y)) = x, \text{snd}(\text{pair}(x,y)) = y \)

Folding variant narrowing

Protocol $P$:
rule Register_pk:
  \[ \text{Fr}(ltk:fr) \rightarrow \ ]
  \[ \text{ltk}(A:pub,ltk:fr), \text{lpk}(A:pub,pkltk:fr) \] 

Variant formulas for Protocol

Constraint Solving:
reduction steps chosen by heuristic (terminating)

Axioms $\psi$:
axiom InEq: "not (\exists x \in x. \text{InEq}(x,x)@i)"

Derived constraint rewriting rules

Security Properties $\varphi$:
lemma SessionKeySecret:
"All A B C key i j.
  Accept(A, B\#C, key)@i &
"

Proof:
solve (Accept(<A,B,X>,key)@i) 
  case Init_1 
  ... 
  qed

Constraint Solving:
reduction steps chosen by heuristic or interactively in GUI
(heuristics might not terminate)

Attack:
displays solved constraint system and visualizes the dependency graph for attack


