Verification of Cryptographic Protocols: Tagging Enforces Termination

Bruno Blanchet[1,2] and Andreas Podelski[2]

[1]Département d'Informatique,
École Normale Supérieure
[2]Max-Planck-Institut für Informatik

April 2003

Motivation

A technique for verifying cryptographic protocols, based on a translation into Horn clauses and a resolution algorithm on the clauses.

- Can handle an unbounded number of sessions
- Can handle a wide range of cryptographic primitives
- Efficient in practice

Experimental results

Pentium III 1 GHz.

Protocol	Time (ms)
Needham-Shroeder public key	51
Denning-Sacco	12
Woo-Lam public key	6
Yahalom	21
Otway-Rees	29
Needham-Shroeder shared key	Non-term.
Woo-Lam shared key [Woo92]	Non-term.
Woo-Lam shared key [Woo97]	8
Woo-Lam shared key corrected	7

Motivation (2)

. . . but does not always terminate !

Our result:

> The algorithm does terminate for *tagged protocols*,
> a wide class of well-designed protocols.

Tagging a protocol

$$\begin{array}{l} \ldots \\ \text{Message 2} : Alice \rightarrow Bob : \{message\}_{key} \\ \ldots \end{array}$$

Tagging a protocol

$$\ldots$$
$$\text{Message 2} : Alice \rightarrow Bob : \{\text{tag}_0, message\}_{key}$$
$$\ldots$$

Label each cryptographic operation with a tag.

- A tag is a constant in the protocol text.

Tagging a protocol

$$\begin{array}{l} \ldots \\ \text{Message 2} : Alice \to Bob : \{\mathtt{tag}_0, message\}_{key} \\ \text{Message 3} : Bob \to Alice : \{\mathtt{tag}_1, message'\}_{key'} \\ \ldots \end{array}$$

Label each cryptographic operation with a tag.

- A tag is a constant in the protocol text.
- Different tags for different messages in the protocol text.
- Different sessions execute the same protocol, so use the same tag

Tagging a protocol

$$
\begin{array}{l}
\ldots \\
\text{Message 2} : Alice \rightarrow Bob : \{\mathtt{tag}_0, message\}_{key} \\
\text{Message 3} : Bob \rightarrow Alice : \{\mathtt{tag}_1, message'\}_{key'} \\
\ldots
\end{array}
$$

Label each cryptographic operation with a tag.

- The sender adds the tag.
- The receiver checks that the tag is correct.

$\Rightarrow$ prevents the receiver from mixing messages 2 and 3.

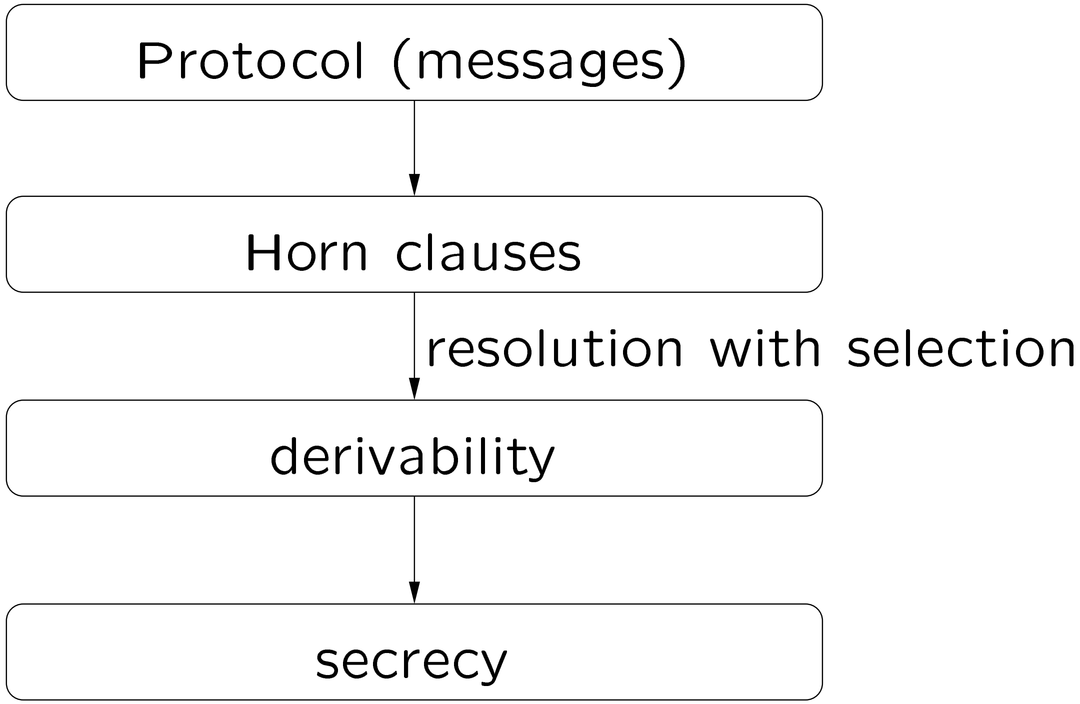
Tagging a protocol

$$\begin{array}{l} \ldots \\ \text{Message 2} : Alice \rightarrow Bob : \{\mathtt{tag}_0, message\}_{key} \\ \text{Message 3} : Bob \rightarrow Alice : \{\mathtt{tag}_1, message'\}_{key'} \\ \ldots \end{array}$$

Label each cryptographic operation with a tag.

- Simple syntactic annotation.
- Retains the intended behavior of the protocol.
- Prevents type flaw attacks.
- Good design practice according to Abadi, Needham [1994]. Used in some real protocols (SSH).

The analysis technique



The Horn clauses

- Computation abilities of the attacker:
  If the attacker has $\mathsf{encrypt}(message, key)$ and key then it has $message$.

$$\mathsf{att}(\mathsf{encrypt}(message, key)) \wedge \mathsf{att}(key) \rightarrow \mathsf{att}(message)$$

- Actions of the protocol participants:
  If the attacker has $message$, it sends it to $Alice$, $Alice$ replies with $message'$, that the attacker intercepts.

$$\mathsf{att}(message) \rightarrow \mathsf{att}(message')$$

Secrecy criterion

$\mathsf{att}(message)$ means "the attacker may have $message$".

$$\boxed{\begin{array}{l}\text{If } \mathsf{att}(message) \text{ is not derivable from the clauses,}\\ \text{then the protocol preserves the secrecy of } message.\end{array}}$$

We determine derivability using a resolution-based algorithm.

Non-termination

Inspired from the Needham-Schroeder shared-key protocol:

$$\text{Message 1. } A \to B : \{N_A\}_K$$
$$\text{Message 2. } B \to A : \{f(N_A)\}_K$$

where N_A is a nonce and f is some function.

Horn clauses:

$$\mathsf{att}(\mathsf{encrypt}(N_A, K))$$
$$\mathsf{att}(\mathsf{encrypt}(x, K)) \to \mathsf{att}(\mathsf{encrypt}(f(x), K))$$

Non-termination (continued)

From

$$\mathsf{att}(\mathsf{encrypt}(N_A, K))$$
$$\mathsf{att}(\mathsf{encrypt}(x, K)) \rightarrow \mathsf{att}(\mathsf{encrypt}(f(x), K))$$

resolution yields:

$$\mathsf{att}(\mathsf{encrypt}(f(N_A), K))$$

Non-termination (continued)

From

$$\mathsf{att}(\mathsf{encrypt}(f(N_A), K))$$
$$\mathsf{att}(\mathsf{encrypt}(x, K)) \rightarrow \mathsf{att}(\mathsf{encrypt}(f(x), K))$$

resolution yields:

$$\mathsf{att}(\mathsf{encrypt}(f(f(N_A)), K))$$

etc.

The algorithm does not terminate !

The same example with tags

$$\text{Message 1. } A \rightarrow B : \{\mathsf{tag}_1, N_A\}_K$$
$$\text{Message 2. } B \rightarrow A : \{\mathsf{tag}_2, f(N_A)\}_K$$

The clauses become:

$$\mathsf{att}(\mathsf{encrypt}((\mathsf{tag}_1, N_A), K))$$
$$\mathsf{att}(\mathsf{encrypt}((\mathsf{tag}_1, x), K)) \rightarrow \mathsf{att}(\mathsf{encrypt}((\mathsf{tag}_2, f(x)), K))$$

and we infer:

$$\mathsf{att}(\mathsf{encrypt}((\mathsf{tag}_2, f(N_A)), K))$$

Resolution stops: tag_1 does not unify with tag_2 !

Main ideas

The algorithm and the proof of termination are technical (see paper).

In this talk:

- Key idea of the algorithm
- Key point of the proof
- What about non-tagged protocols ?

A naive algorithm never terminates

For instance, the following clause

$$\mathsf{att}(x) \wedge \mathsf{att}(y) \rightarrow \mathsf{att}(\mathsf{encrypt}(x, y))$$

resolves with itself:

$$\frac{\begin{array}{l}\mathsf{att}(x) \wedge \mathsf{att}(y_1) \rightarrow \mathsf{att}(\mathsf{encrypt}(x, y_1)) \\ \qquad\qquad\qquad\qquad \mathsf{att}(x') \wedge \mathsf{att}(y_2) \rightarrow \mathsf{att}(\mathsf{encrypt}(x', y_2))\end{array}}{\mathsf{att}(x) \wedge \mathsf{att}(y_1) \qquad\qquad \wedge\, \mathsf{att}(y_2) \rightarrow \mathsf{att}(\mathsf{encrypt}(\mathsf{encrypt}(x, y_1), y_2))}$$

yielding

$$\mathsf{att}(x) \wedge \mathsf{att}(y_1) \wedge \ldots \wedge \mathsf{att}(y_n) \rightarrow \mathsf{att}(\mathsf{encrypt}(\ldots \mathsf{encrypt}(x, y_1), \ldots, y_n))$$

so unconstrained resolution never terminates.

$\Rightarrow$ restrict allowed resolution steps, while retaining completeness.

Key idea of the algorithm

Forbid resolution on terms that are variables:

$$\frac{H \rightarrow \mathsf{att}(M) \qquad \mathsf{att}(M') \wedge H' \rightarrow C'}{\sigma_u H \wedge \sigma_u H' \rightarrow \sigma_u C'}$$

resolution rule applied only when both M and M' are non-variable terms

Then M and M' must have the same root symbol.

For tagged protocols, M and M' must, in addition, have the same tag.

$\Rightarrow$ strongly restricts possible resolution steps.

Key point of the proof

For all generated clauses, the terms are
more general than subterms of the intended messages
of the protocol (messages of an attack-free execution).

Example:

$$\text{Message 1. } A \to B : \{\mathsf{tag}_1, N_A\}_K$$
$$\text{Message 2. } B \to A : \{\mathsf{tag}_2, f(N_A)\}_K$$

Intended messages: $\mathsf{encrypt}((\mathsf{tag}_1, N_A), K)$, $\mathsf{encrypt}((\mathsf{tag}_2, f(N_A)), K)$
Subterms: intended messages and tag_1, tag_2, N_A, $f(N_A)$, K.

Key point of the proof

Property on previous slide ("all generated terms more general than finite set of subterms of intended messages") is interesting:

- Not true in general for non-tagged protocols.

(see example $\mathsf{encrypt}(f^n(N_A), K)$)

- Does not mean that there are no attacks.

(Terms may have instances other than the one in the finite set of subterms of intended messages.)

- Key step towards termination:

Bounds the terms in the clauses.
(We still have to bound the number of hypotheses.)

Size decrease

To obtain termination, we show:

> The size of an instance of the clauses in the subterms of intended messages decreases.

The size of the clauses themselves does not decrease, since unification can instantiate terms.

Experiments with tagging

Pentium III 1 GHz.

Protocol	Tagging	Time (ms)
Needham-Shroeder shared key	Not	Non-term.
Needham-Shroeder shared key	Tagged	27
Woo-Lam shared key [Woo92]	Not	Non-term.
Woo-Lam shared key	Tagged	7
Woo-Lam shared key [Woo97]	Not	8
Woo-Lam shared key corrected	Tagged	7
Needham-Shroeder public key	Not	51
Denning-Sacco	Not	12
Woo-Lam public key	Not	6
Yahalom	Not	21
Otway-Rees	Not	29

Implicit tags

The Woo-Lam public key protocol (1997):

Message 1.	$A \to B:$	A
Message 2.	$B \to A:$	N
Message 3.	$A \to B:$	$\{A, B, N\}_{sk_A}^{\text{ar. } 3}$
Message 4.	$B \to S:$	A
Message 5.	$S \to B:$	$A, \{A, pk_A\}_{sk_S}^{\text{ar. } 2}$

The arity (in this example) can be used to distinguish different encryptions

$\Rightarrow$ We also prove termination in this case

Implicit tagging

Pentium III 1 GHz.

Protocol	Tagging	Time (ms)
Needham-Shroeder shared key	Not	Non-term.
Needham-Shroeder shared key	Tagged	27
Woo-Lam shared key [Woo92]	Not	Non-term.
Woo-Lam shared key	Tagged	7
Woo-Lam shared key [Woo97]	Not	8
Woo-Lam shared key corrected	Tagged	7
Needham-Shroeder public key	Not	51
Denning-Sacco	Implicit	12
Woo-Lam public key	Implicit	6
Yahalom	Not	21
Otway-Rees	Not	29

Conclusion

- Tag your protocol!
  You will have the guarantee of termination.

- Implicit tagging partly explains why the algorithm often terminated even before adding tags.

- Is there a larger class on which the algorithm terminates ?
  (. . . a natural one ?)