Formal verification of security protocols: Verifpal

Stéphanie Delaune and Clément Hérouard
Univ Rennes, CNRS, IRISA, Spicy team

CORGIS - January 9, 2024

Formal verification of cryptographic protocols

Security protocol design is critical and error-prone as illustrated by many attacks: FREAK, Logjam, ...

Use formal methods to improve confidence:

- prove the absence of attacks under certain assumptions; or
- identify weaknesses.

Many verification tools already exist:

- Proverif, CryptoVerif, Tamarin, EasyCrypt, DeepSec, Squirrel, ... and **Verifpal**.

Two famous examples of logical attacks

An authentication flaw on the Needham Schroeder protocol

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS protocol (1978)

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B, B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS-Lowe protocol (1995)

Two famous examples of logical attacks

An authentication flaw on the Needham Schroeder protocol

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS protocol (1978)

$$A \to B : \{A, N_A\}_{\mathrm{pub}(B)}$$
$$B \to A : \{N_A, N_B, B\}_{\mathrm{pub}(A)}$$
$$A \to B : \{N_B\}_{\mathrm{pub}(B)}$$

NS-Lowe protocol (1995)

FREAK attack by Barghavan et al. (2015)

A logical flaw that allows a *man-in-the-middle* attacker to downgrade connections from 'strong' RSA to 'export grade' RSA.

$\longrightarrow$ websites affected by the vulnerability included those from the US federal government

Two main families of models

Symbolic models	**Computational models**
[Dolev & Yao, 81]	[Goldwasser & Micali, 84]

Two main families of models

Symbolic models [Dolev & Yao, 81]	**Computational models** [Goldwasser & Micali, 84]
Messages are terms.	Messages are bitstrings.
What the attacker can do.	What the attacker can **not** do. Everything else is allowed![1]
Weaker guarantees.	Stronger guarantees.
Amenable to automation.	Harder to automate.
e.g. Proverif, Tamarin, Verifpal	e.g. CryptoVerif

[1] The attacker is a probabilistic polynomial-time Turing machine.

Two main families of security properties

- Reachability properties, *e.g.* weak secrecy, authentication, ...

  Is it possible to reach a bad state, e.g. a state in which the secret would be deducible?

- Equivalence properties, *e.g.* anonymity, unlinkability, strong form of secrecy, ...

  Is the attacker able to distinguish between two situations?

$\overset{?}{\approx}$

$\longrightarrow$ different notions of equivalence actually exist.

Brief comparison - symbolic verification tools

	DeepSec/Akiss	ProVerif/GSverif	Tamarin	**Verifpal**
unbounded traces	✗	✓	✓	?
automation	✓	↗	↗	?
custom primitives	✓	✓	✓	?
equivalence properties	✓	↗	↗	?
user interface	✓	✓	✓	?
user manual	✓	✓	✓	?

What about Verifpal ?

Brief comparison - symbolic verification tools

	DeepSec/Akiss	ProVerif/GSverif	Tamarin	**Verifpal**
unbounded traces	✗	✓	✓	?
automation	✓	↗	↗	?
custom primitives	✓	✓	✓	✗
equivalence properties	✓	↗	↗	?
user interface	✓	✓	✓	✓
user manual	✓	✓	✓	✓

What about Verifpal ?

Why Verifpal?

Main developer: Nadim Kobeissi - The project starts in 2019.

`https://verifpal.com`

> *Despite this increase in the usage of formal verification tools, and despite the success obtained with this approach, automated formal verification technology remains unused outside certain specific realm of academia.*

Simplifying protocol analysis with Verifpal

1. an intuitive language for modeling protocols;
2. modeling that avoid user error;
3. easy to understand analysis output;
4. compatibility with the Coq theorem prover (only regarding the passive attacker).

Modeling using Verifpal

Modeling a simple example (1/3)

```
attacker[active]

principal Alice[
        generates a
        ga = G^a
]

principal Bob[
        generates b
        gb = G^b
]

Alice -> Bob: [ga]
Bob -> Alice: [gb]
```

- passive or active attacker;
- generation of nonce;
- primitives are built-in;
- guarded constants (i.e. [_]): an atttacker can still read it, but can not tamper with it.

Modeling a simple example (2/3)

```
principal Alice[
        generates m
        e = PKE_ENC(gb, m)
        h = MAC(gb^a, e)
]

Alice -> Bob: e, h

principal Bob[
        _ = ASSERT(MAC(ga^b, e), h)?
        d = PKE_DEC(b, e)
]
```

- checked primitives: adding a ? after a primitive means that the execution will abort in case of a failure

Modeling a simple example (3/3)

```
queries[
        confidentiality? m
        authentication? Alice -> Bob: e
]
```

1. confidentiality: Is m deducible by the attacker?
   $\longrightarrow$ phases can be used to model forward secrecy;
2. authentication: Is e received/used by Bob as it was sent by Alice?
   $\longrightarrow$ rely on the notion of checked primitives;
3. freshness: ??
4. unlinkability: "we note that unlinkability queries are especially experimental"

$\longrightarrow$ The semantics is not given but we can assume (?) that there is no fancy things there.

Verifying using Verifpal

Verifying the simple example

```
macdelaune2:verifpal-examples delaune$
macdelaune2:verifpal-examples delaune$ verifpal verify pke.vp
Verifpal 0.26.1 - https://verifpal.com
  Warning • Verifpal is Beta software.
 Verifpal • Parsing model 'pke.vp'...
 Verifpal • Verification initiated for 'pke.vp' at 10:50:16 AM.
     Info • Attacker is configured as active.
     Info • Running at phase 0.
 Analysis • Constructed skeleton ASSERT(MAC(G^nil, PKE_ENC(G^nil, nil)), MAC(G^nil, PKE_ENC(G^nil, nil))) based on ASSERT(MAC(G^a^b, PKE_ENC(G^b, m)), MAC(G^b^a, PKE_ENC(G^b, m)))?.
 Analysis • Constructed skeleton PKE_DEC(nil, PKE_ENC(G^nil, nil)) based on PKE_DEC(b, PKE_ENC(G^b, m)).
Deduction • Output of ASSERT(MAC(G^a^b, PKE_ENC(G^b, m)), MAC(G^b^a, PKE_ENC(G^b, m)))? obtained by reconstructing with MAC(G^a^b, PKE_ENC(G^b, m)), MAC(G^b^a, PKE_ENC(G^b, m)).
Deduction • Output of PKE_ENC(G^nil, nil) obtained by decomposing PKE_DEC(nil, PKE_ENC(G^nil, nil)) with nil.
 Analysis • Initializing Stage 1 mutation map for Alice...
 Analysis • Initializing Stage 1 mutation map for Bob...
Deduction • G^nil obtained by reconstructing with nil. (Analysis 2)
 Analysis • Initializing Stage 2 mutation map for Alice...
 Analysis • Initializing Stage 2 mutation map for Bob...
 Analysis • Initializing Stage 3 mutation map for Alice...
 Analysis • Initializing Stage 3 mutation map for Bob...
Deduction • Output of PKE_ENC(G^a, nil) obtained by reconstructing with G^a, nil. (Analysis 4)
Deduction • Output of PKE_ENC(G^b, nil) obtained by reconstructing with G^b, nil. (Analysis 6)
 Analysis • Initializing Stage 5 mutation map for Alice...
 Analysis • Initializing Stage 4 mutation map for Alice...
 Analysis • Initializing Stage 5 mutation map for Bob...
 Analysis • Initializing Stage 4 mutation map for Bob...
 Analysis • Initializing Stage 6 mutation map for Alice...
 Analysis • Initializing Stage 6 mutation map for Bob...
 Stage 6, Analysis 29...

 Verifpal • Verification completed for 'pke.vp' at 10:50:16 AM.
 Verifpal • All queries pass.

 Verifpal • Thank you for using Verifpal.
```

Some variations on the simple example

Some variations on the simple example

Authentication fails when:

1. removing ?
2. when adding a dummy line, e.g. dummy = CONCAT(e,e) or swapping the two lines of Bob at the end of the role
3. removing [_] around gb

Confidentiality fails when:

1. removing [_] around ga

Analysis in Verifpal

"Verifpal's active attacker analysis methodology follows a simple set of procedures and algorithms"

1. gather values;
2. insert learned values into attacker state ($\mathcal{V}_{\mathcal{A}}$);
3. apply transformations: e.g. deconstruct on $\mathcal{V}_{\mathcal{A}}$, construct within $\mathcal{V}_{\mathcal{P}}$ from $\mathcal{V}_{\mathcal{A}}$;
4. prepare mutation for next session:
5. iterate across protocol mutations.

Analysis in Verifpal

"Verifpal's active attacker analysis methodology follows a simple set of procedures and algorithms"

1. gather values;
2. insert learned values into attacker state ($\mathcal{V}_{\mathcal{A}}$);
3. apply transformations: e.g. deconstruct on $\mathcal{V}_{\mathcal{A}}$, construct within $\mathcal{V}_{\mathcal{P}}$ from $\mathcal{V}_{\mathcal{A}}$;
4. prepare mutation for next session:
5. iterate across protocol mutations.

What is the state space explored exactly ? Is termination guaranteed?

Verifpal is a tool for finding attacks only

ProVerif, Tamarin, and Deepsec are able to find attacks but they achieve also some form of completeness

$\longrightarrow$ **no completeness result in Verifpal.** The tool can miss logical attacks !!

Verifpal is a tool for finding attacks only

ProVerif, Tamarin, and Deepsec are able to find attacks but they achieve also some form of completeness

$\longrightarrow$ **no completeness result in Verifpal.** The tool can miss logical attacks !!

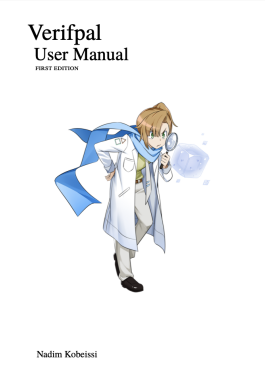
> *"Formally, Verifpal is unable to claim that it never misses an attack in any model that can be expressed within its language."*

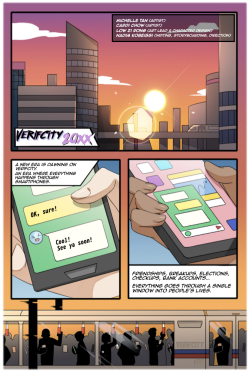
> *"However, our hope is that Verifpal would not miss attacks affecting models of, or resembling, real-world protocols."*

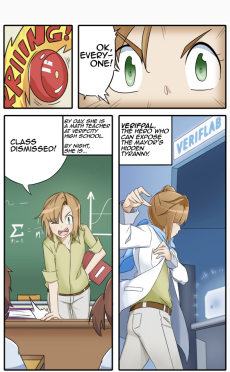
Advertisement

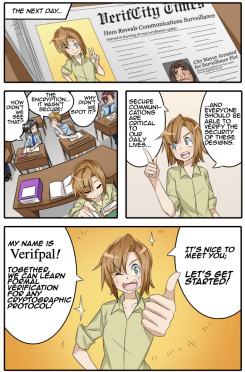
A nice user manual with beautiful illustations !

⟶ the syntax is given but no formal semantics regarding protocols, security properties.









Official Demo

https://www.youtube.com/watch?v=JrJt6hOAQlQ

Unofficial Demo

Needham-Schroeder's Protocol (1978)

- $A \rightarrow B: \{A, N_a\}_{\mathrm{pub}(B)}$
  $B \rightarrow A: \{N_a, N_b\}_{\mathrm{pub}(A)}$
  $A \rightarrow B: \{N_b\}_{\mathrm{pub}(B)}$

Needham-Schroeder's Protocol (1978)

- $$\begin{array}{lcll} A & \rightarrow & B: & \{A, N_a\}_{\mathrm{pub}(B)} \\ B & \rightarrow & A: & \{N_a, N_b\}_{\mathrm{pub}(A)} \\ A & \rightarrow & B: & \{N_b\}_{\mathrm{pub}(B)} \end{array}$$

Needham-Schroeder's Protocol (1978)

$$
\begin{array}{lcll}
 & A & \rightarrow & B: \quad \{A, N_a\}_{\mathrm{pub}(B)} \\
 & B & \rightarrow & A: \quad \{N_a, N_b\}_{\mathrm{pub}(A)} \\
\bullet & A & \rightarrow & B: \quad \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Needham-Schroeder's Protocol (1978)

$$
\begin{array}{lcll}
A & \rightarrow & B: & \{A, N_a\}_{\mathrm{pub}(B)} \\
B & \rightarrow & A: & \{N_a, N_b\}_{\mathrm{pub}(A)} \\
A & \rightarrow & B: & \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Questions

- Is N_b secret between A and B ?
- When B receives $\{N_b\}_{\mathrm{pub}(B)}$, does this message really comes from A ?

Needham-Schroeder's Protocol (1978)

$$
\begin{array}{lcll}
A & \rightarrow & B: & \{A, N_a\}_{\mathrm{pub}(B)} \\
B & \rightarrow & A: & \{N_a, N_b\}_{\mathrm{pub}(A)} \\
A & \rightarrow & B: & \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Questions

- Is N_b secret between A and B ?
- When B receives $\{N_b\}_{\mathrm{pub}(B)}$, does this message really comes from A ?

Man in the middle attack found by G. Lowe (1995)

Agent A

Intrus I

Agent B

Attack

- involving 2 sessions in parallel,
- an honest agent has to initiate a session with I.

$$
\begin{aligned}
A \rightarrow B &: \{A, N_a\}_{pub(B)} \\
B \rightarrow A &: \{N_a, N_b\}_{pub(A)} \\
A \rightarrow B &: \{N_b\}_{pub(B)}
\end{aligned}
$$

Man in the middle attack found by G. Lowe (1995)

Agent A

$\xrightarrow{\{A, N_a\}_{\mathrm{pub}(I)}}$

Intrus I

$\xrightarrow{\{A, N_a\}_{\mathrm{pub}(B)}}$

Agent B

Attack

- involving 2 sessions in parallel,
- an honest agent has to initiate a session with I.

$$
\begin{array}{lll}
\mathrm{A} \rightarrow \mathrm{B} & : & \{A, N_a\}_{\mathrm{pub}(B)} \\
\mathrm{B} \rightarrow \mathrm{A} & : & \{N_a, N_b\}_{\mathrm{pub}(A)} \\
\mathrm{A} \rightarrow \mathrm{B} & : & \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Man in the middle attack found by G. Lowe (1995)

$\xrightarrow{\{A, N_a\}_{\mathrm{pub}(I)}}$

$\xleftarrow{\{N_a, N_b\}_{\mathrm{pub}(A)}}$

$\xrightarrow{\{A, N_a\}_{\mathrm{pub}(B)}}$

$\xleftarrow{\{N_a, N_b\}_{\mathrm{pub}(A)}}$

Agent A

Intrus I

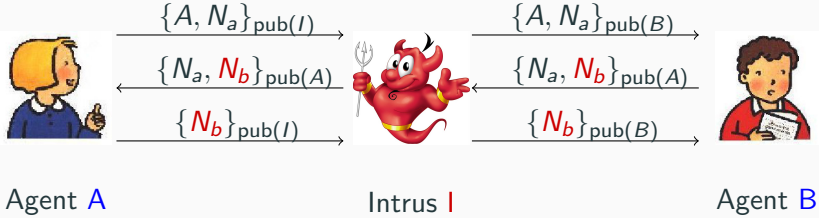
Agent B

Attack

- involving 2 sessions in parallel,
- an honest agent has to initiate a session with I.

$$
\begin{array}{lcl}
\mathrm{A} \rightarrow \mathrm{B} & : & \{A, N_a\}_{\mathrm{pub}(B)} \\
\mathrm{B} \rightarrow \mathrm{A} & : & \{N_a, N_b\}_{\mathrm{pub}(A)} \\
\mathrm{A} \rightarrow \mathrm{B} & : & \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Man in the middle attack found by G. Lowe (1995)



Attack

- involving 2 sessions in parallel,
- an honest agent has to initiate a session with I.

$$
\begin{array}{lll}
\mathrm{A} \rightarrow \mathrm{B} & : & \{A, N_a\}_{\mathrm{pub}(B)} \\
\mathrm{B} \rightarrow \mathrm{A} & : & \{N_a, N_b\}_{\mathrm{pub}(A)} \\
\mathrm{A} \rightarrow \mathrm{B} & : & \{N_b\}_{\mathrm{pub}(B)}
\end{array}
$$

Conclusion - Takeaways

What about using Verifpal to perform a security analysis?

Advantages:

A nice user manual with illustrations and a nice user interface.

Warning:

- not so easy to install on a Linux;
- reachability properties only;
  $\longrightarrow$ *regarding unlinkability, the author himself is not confident!*
- do not cover the same spectrum as other tools
  $\longrightarrow$ *a successful analysis in ProVerif or Tamarin provides more guarantee.*

Questions ?