frama C

Software Analyzers

# From Manual to LLM-Assisted Formal Verification of a PKCS#1 Signature Parser

Martin Hána[1] Nikolai Kosmatov[2] Virgile Prevosto[3] Julien Signoles[3]

2026-10-28 @ AISoLA'26

[1]Thales Cybersecurity and Digital Identity

[2]Thales Research & Technology, cortAIx Labs

[3]Université Paris-Saclay, CEA, List

cea list

université PARIS-SACLAY

THALES

FRANCE 2030 PROGRAMME DE RECHERCHE CYBERSÉCURITÉ

# Outline

Context
- Message Parsers
- Frama-C Verification Framework

PKCS#1v1.5 and Signature Forgery

LLM in Formal Verification Cycle
- Methodology
- Evaluation

Conclusion

## Motivation

- applying formal methods to solve difficult or long-living security problems in academic and industrial context
- secure parsing for messages based on the TLV format
- investigate how LLM can help formal specification and verification

## Use case: Bleichenbacher's PKCS#1v1.5 signature forgery

- long-living, affecting many X.509 certificates and open-source libraries
  - Initial Bleichenbacher's attack: 2006
  - A variant was still found in 2021
- root cause: wrong (too lax) parsing

- A Framework for modular analysis of C code.
- Released under LGPL (v33.0 in July 2026)
- `https://frama-c.com/`
- Extensible platform
- ACSL (ANSI/ISO C Specification language)



# ACSL

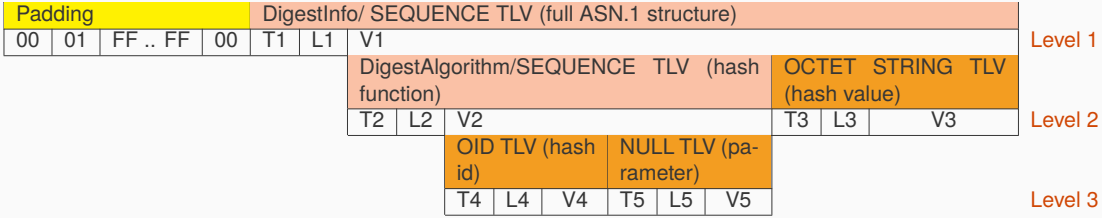
```
1  /*@
2   requires \valid(state + (0 .. n - 1));
3   requires
4     \forall integer i;
5     0 <= i < n ==> elem_ok(state[i]);
6   assigns state[0 .. n];
7   ensures computation_ok(state);
8  */
9  void f(char* state, size_t n) {
10 ...
11 //@ assert some_property(state);
12 ...
13 }
```

- Based on function contracts
- First-order logic formulas
- Built-ins predicates for basic C concepts
- Specify properties that are supposed to hold at specific program points

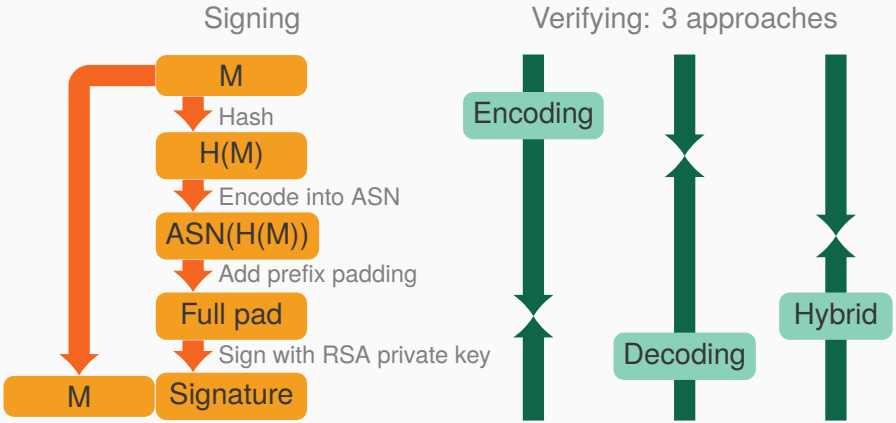
# PKCS#1 v1.5 Signature Format

## ASN.1 standard: Tag/Length/Value (TLV) format

- Multiple nested TLVs (3 levels)
- Stores hash function (id and parameter) and hash value



# PKCS#1v1.5 Signature Processing




Focus on decoding approach

- more risky security-wise
- lowest memory consumption

# Bleichenbacher's Attack (2006)

## 2 necessary conditions

- Signature parser ignores a sufficient number of bytes
- Signature is verified using 3 as public exponent

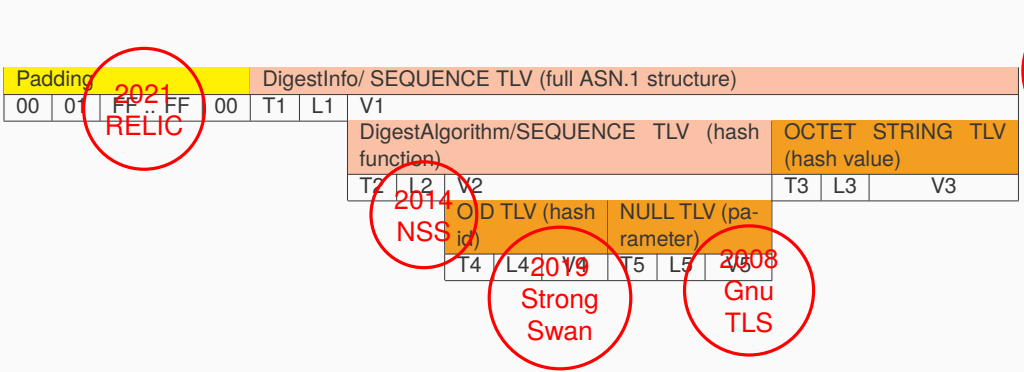
## Practicality of the attack

- Allow signature forgery, e.g. of X.509 certificates signing
- Affected public key still exist in Ubuntu ca-certificates

[1] Finney, H.: Bleichenbacher's RSA signature forgery based on implementation error (Crypto 2006)

[2] Yahyazadeh, M. et al.: Morpheus: Bringing The (PKCS) One To Meet the Oracle (CCS 2021)

# Ignored Bytes: Known Attacks



# Experimental Setup

## Code

- Homemade PKCS#1v1.5 parser in C.
- Excludes RSA primitive operations.
- Already formally verified with Frama-C/Wp [1].

## LLM

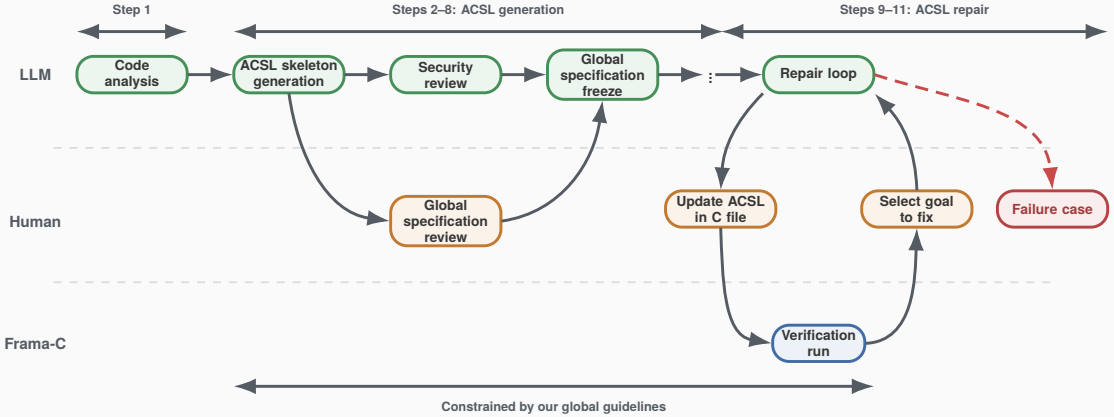
- ChatGPT 5.1 to 5.4
- Plus subscription tier
- no agentic setup

## Targeted Properties

- Parser accepts a message only if it has checked all bytes.
- No well-formed message is rejected.
- No RTE can occur.

[1] Hána, M. et al.: Formal Verification of PKCS#1 Signature Parser using Frama-C (iFM 2025)

# Process


Step 1
Steps 2–8: ACSL generation
Steps 9–11: ACSL repair
LLM
Code analysis
ACSL skeleton generation
Security review
Global specification freeze
Repair loop
Human
Global specification review
Update ACSL in C file
Select goal to fix
Failure case
Frama-C
Verification run
Constrained by our global guidelines

# Code Analysis and Summarization

## Objective

- Identify entry points and verification perimeter.
- Extract call graph.
- Summarize informal properties.
- Retain most important information in session context.

## Sample LLM output

So the strongest informal statement is: On nominal return, the parser has established that the provided block is a well-formed PKCS#1 v1.5 encoded digest container for the caller-expected hash algorithm, and that its embedded digest matches the recomputed digest of the supplied input message.

# Global Alignment Constraints

## Objective

- Minimize the risk of introducing incoherences.
- Facilitate the auditability of the generated specification.
- Freeze global specification once it is deemed satisfactory.

## Sample Guideline

7. All hypotheses should be included into the model only EITHER as (a) an entry point precondition OR (b) a stub function postcondition. For both cases, be careful not to include logical contradictions and inconsistencies! ... If you modify some of the existing hypotheses, raise a warning to me using the message “MODIFY MODEL HYPOTHESES - MANUAL REVIEW MANDATORY”. ...
9. Don’t use ACSL admit clause.

# Global ACSL Specification

## Steps

- Generate the actual specification to be formally verified.
- Cross-evaluation by the LLM with specific prompt.
- Very careful human review.

## Sample Predicate

```
// Constructed TLV for AlgorithmIdentifier VALUE:
//  OID || NULL
predicate CTLV_digalg(integer size, u1 *p, integer alg)
  =
  // structure and tags
  SupportedHashAlg(alg) &&
  (\let t1_off = 0;
   p[t1_off] == TAG_OID &&
   (\let t2_off = t1_off + p[t1_off+1] + LEN_TAGLEN;
    p[t2_off] == TAG_NULL &&
    // DigestAlgorithm size consistency
    size == t2_off + p[t2_off+1] + LEN_TAGLEN &&
    // semantic properties of primitive children
    OIDValueMatches(alg, &p[t1_off + LEN_TAGLEN],
                    p[t1_off+1])
    &&
    p[t2_off+1] == ASN_LEN_NULL));
```

# Repair Loop and Supplementary Annotations

## Steps

- Ask to generate auxiliary annotations.
- Involves adding annotations to prove absence of RTE.
- And few-shot examples to deal with potential Smoke-tests failures
- Attempt to prove with Wp the global specification.
- Manual selection of the annotations to fix

## Sample output

The failing VC says that [...], Wp cannot prove the current precondition:

```
requires 0<=*num_read<=num_to_read;
```

But this precondition is actually stronger than what the C code needs. [...]
Minimal fix: We should weaken the precondition of `read_one_tlv` [...]

- keep `0 <= *num_read`
- keep `*num_read<=INT_MAX-LEN_TAGLEN` to avoid overflow [...]
- require msg validity only in the branch where the header is actually read

# Bug Detection Capabilities

- Process replayed starting from known vulnerable variation of the parser.
- Initial generated global specification sometimes reflects the same bug as the code
- LLM self-assessment of its global specification usually effective in catching the issue, but not always.
- Human expertise still needed.

# Comparison with Handcrafted Specification

- Generated global specification improved auditability in some parts of the specification
- LLM generated fewer lines of annotations (226 vs. 364)
- Also smaller global specification (99 vs. 227)
- Proof Obligations more easily handled by the provers (1min 16s vs. 2min 57s)
- Human effort substantially less (2.5pm vs 6pm)

## Caveat

Manual specification was written with reusability across TLV formats in mind, LLM was not instructed to do that.

## Key Take-aways

- Possible to use a LLM for non-trivial formal security verification.
- LLM use ease verification task.
- Human expertise still needed for validating global specification, and guiding LLM in the repair loop

## Research directions

- Extend approach to other parsers
- Use other input than the code for global specification
- More precise token cost evaluation, comparison with other LLMs
- Agentic integration (including for generating proof scripts)