

IETF 123  
CBOR side meeting  
July 22<sup>nd</sup>, 2025

# A Formal Model and Verified Implementation for CBOR, CDDL and COSE

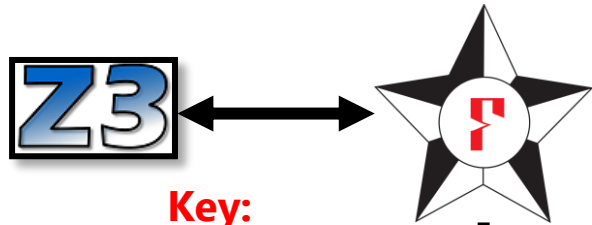
Tahina Ramananandro  
[taramana@microsoft.com](mailto:taramana@microsoft.com)

Microsoft Research  
RiSE – Research in Software Engineering

# Overview

- EverCBOR: Verified parser and serializer for CBOR
  - Written and proven correct in F\*
  - Extracts to C or Rust
- EverCDDL: Verified code generator for CDDL
  - Written and proven correct in F\*
  - User provides a CDDL data format description
  - EverCDDL generates C or Rust:
    - Datatype
    - Parser
    - Serializer
  - Enough for COSE and other protocols
- [ACM CCS 2025](#) (accepted for publication, to appear)
- Open-source: <https://github.com/project-everest/everparse>

# Our Vehicle: F\*, a proof-oriented language for verified low-level programming



**Key:**  
**Automatic**  
**verification**

**F\* implementation**  
and **specification**

**Karamel compiler to**  
**C/Wasm**  
([ICFP 2017](#), [S&P 2019](#))  
Recently extended  
to Rust



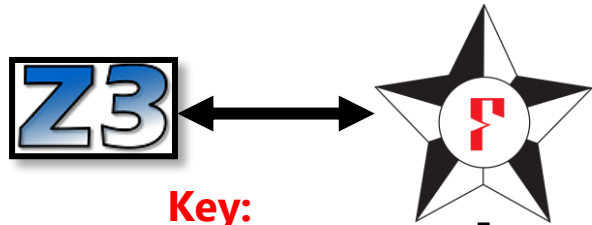
**Efficient C/Wasm/Rust**  
**implementation**

Verification imposes no  
runtime performance  
overhead

```
let multiply_by_9 (a:uint32) : Pure uint32
  (requires 9 * a <= MAX_UINT_32)
  (ensures λ result -> result == 9 * a)
=
  let b = a << 3ul in
  a + b
```

```
uint32_t multiply_by_9(uint32_t a)
{
  uint32_t b = a << (uint32_t)3;
  return a + b;
}
```

# Our Vehicle: F\*, a proof-oriented language for verified low-level programming



**Key:**  
**Automatic**  
**verification**

**Implementation** in the  
Pulse DSL embedded in F\*  
and **specification**  
using separation logic  
([PLDI 2025](#))

```
fn inc (r: ref uint32_t)
  (requires r |-> 'vr * pure (vr + 2 <= MAX_UINT_32))
  (ensures r |-> ('vr + 2))
{
  let v = !r;
  r := v + 2
}
```

**Karamel compiler to**  
**C/Wasm**  
([ICFP 2017](#), [S&P 2019](#))  
Recently extended  
to Rust



**Efficient C/Wasm/Rust**  
**implementation**  
Verification imposes no  
runtime performance  
overhead

```
void inc(uint32_t *r)
{
  uint32_t v = *r;
  *r = v + 2;
}
```

# Proof-Oriented Programming At Scale

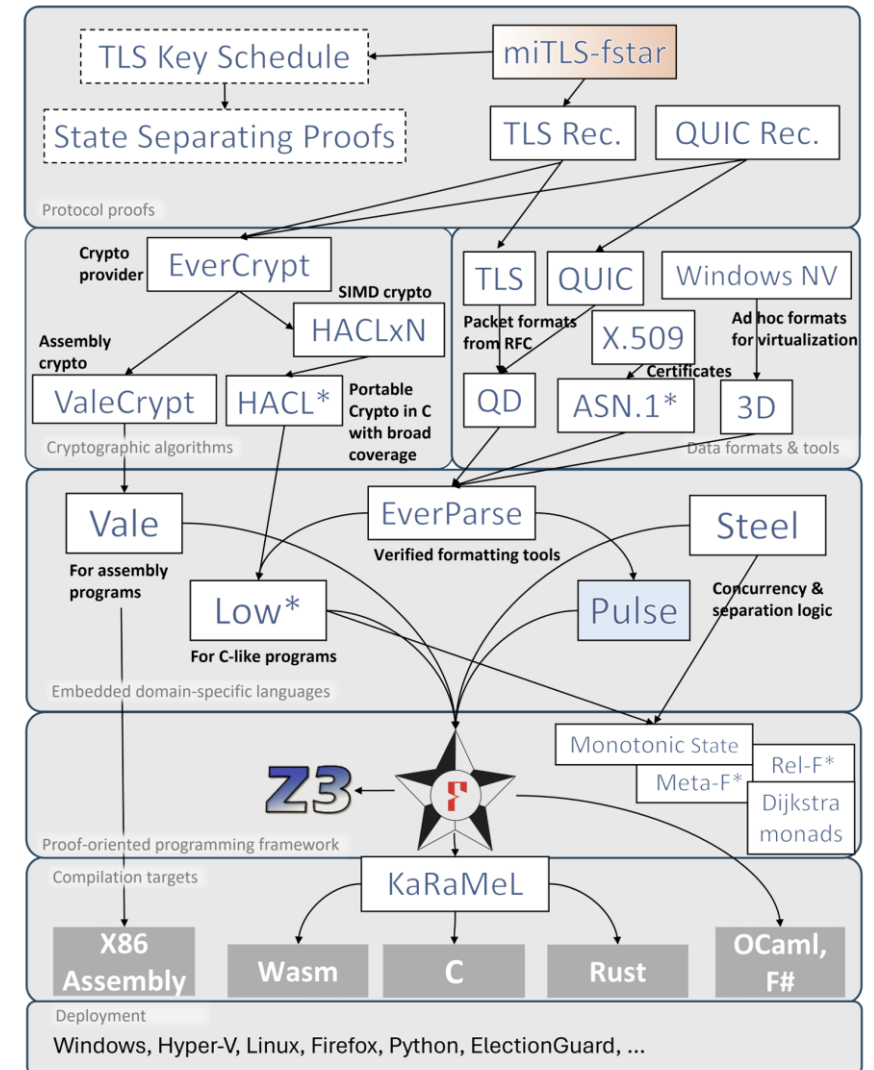
## Project Everest (2016-2021)

- ~1 million lines of F\* code
- Cryptography, verified parsers, ...
- Deployed in Linux, Firefox, Python, Windows, Azure, ...
- <https://project-everest.github.io/>

Everest offspring (including F\*, EverParse) grown up as active standalone projects

EverCBOR, EverCDDL are part of EverParse

- Broader talk at UFMRG on Thursday 7/24



# EverCBOR Formal Guarantees

- Specification
  - Non-ambiguity: serialization parses back to the same CBOR object
  - CBOR§4.2 is non-malleable: unique binary representation
- Implementation
  - Memory-safe, absence of panics or integer overflows/underflows
  - Functionally correct wrt. specification

# EverCBOR implementation

- Definite-length
- Current status
  - Deterministic encoding (CBOR\$4.2)
  - No support for floating-point
- No heap allocation
- API: constructor and destructor functions
  - Constructor for maps takes mutable map entry array, reorders it wrt. deterministic encoding *before* serialization
  - Sorting at the level of objects instead of bytes

# EverCBOR implementation

- Validator
  - CBOR§4.2 validation in constant stack space
  - Definite-length validation in constant stack space if no maps in map keys
  - Unbounded stack in general
- Parser provides iterators for arrays and maps
- Serializer
  - Stack consumption linear in the nesting level
  - No further need for map sorting, since ordering maintained within the item

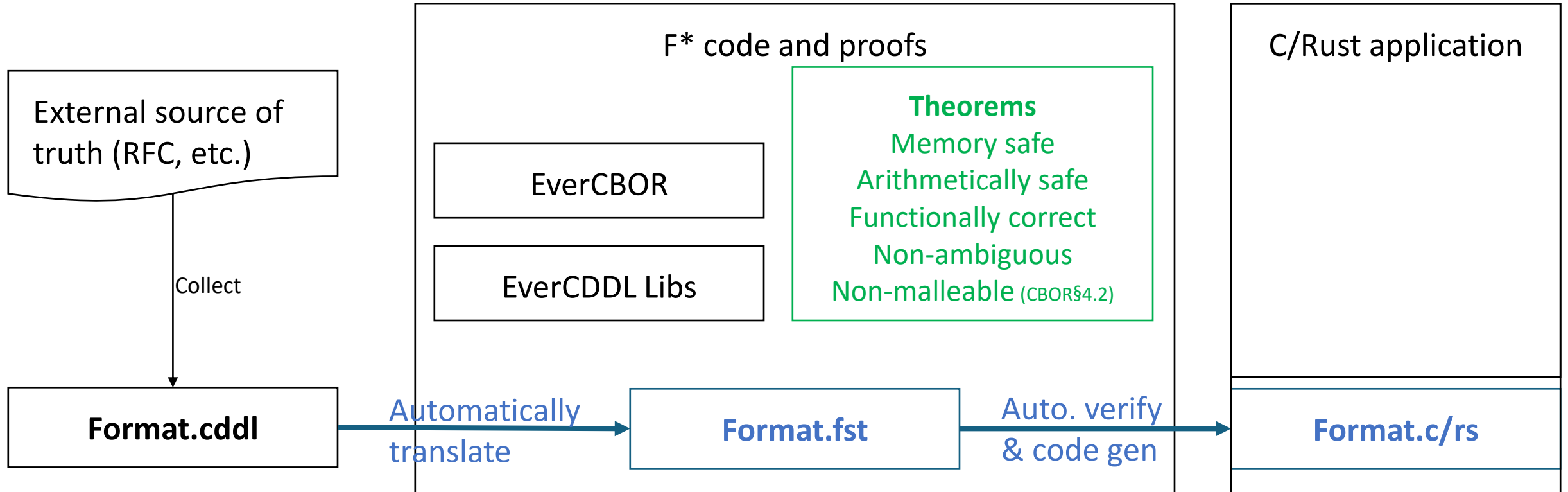


# EverCDDL

## 1. Author spec

## 2. Proof-checking & codegen

## 3. Integrate



# EverCDDL Formal Guarantees

- Specification
  - Non-ambiguity: serialization parses back to the same value
  - Unique CBOR object representation
  - Translates to non-malleability (unique binary representation) with CBOR\$4.2
- Generated code
  - Memory-safe, absence of panics or integer overflows/underflows
  - Functionally correct

# EverCDDL Implementation

- Generates:
  - program datatypes
  - parsers from CBOR objects to program datatype values
  - serializers from program datatype values directly to bytes
  - Currently serializes to CBOR§4.2
- Produces correct-by-construction F\* code that extracts to C or Rust
- Enough for most COSE data structures from RFC 9052

# EverCDDL: Validation and Parsing

Given a CDDL description:

- Validation: is a CBOR object valid with respect to that description?
- Parsing:
  - Generate a C/Rust datatype
  - Given a CBOR object valid wrt. that description, extract its fields into a value of that datatype
  - Generate iterators for arrays/tables
  - No heap allocation

# EverCDDL: Serialization

Given a CDDL description:

- Given a value of the datatype generated by EverCDDL, write the deterministic encoding of the CBOR object that will parse back to that value
- Gracefully fails if it is not serializable:
  - Array/map sizes must be less than  $2^{64}$
  - Integer range / string size constraints must hold
  - A table must not contain entries whose keys match previous items
  - The output buffer must be large enough

# EverCDDL: Serialization

- No additional heap allocation
  - Additional stack usage linear in the size of the spec, not the input
  - Write array/map entries, then shift to write the size
- For each map entry append, insertion sort with 1 swap
  - Establishes and maintains the order
  - Automatically detects duplicates

# EverCBOR, EverCDDL Benchmarks

	EverCDDL		QCBOR		TinyCBOR	
	V/P	S	V/P	S	V/P	S
Rec ( $\mu s$ )	3.33	.57	1.91	.23	3.78	.29
Map ( $\mu s$ )	138		282		306	
Arr (s)	2.67/4.92	2.06	2.92/2.91	0.75	2.68/2.68	1.23

Table 1. Synthetic benchmarks for EverCDDL, QCBOR and TinyCBOR. Values are time (for Rec, for **V**alidation plus **P**arsing, or **S**erialization), lookup time (for Map), or time (for Arr). We distinguish validation from parsing in Arr, since iteration is involved.

- Results on Intel Xeon W-2255 with gcc 11.4 -O3
- Also: We pass the deterministic encoding test cases

# EverCDDL for COSE

	C API & OpenSSL	Pulse API & HACL★
COSE_sign	39.0 $\mu$ s/iter	53.3 $\mu$ s/iter
COSE_verify	99.6 $\mu$ s/iter	58.2 $\mu$ s/iter
Ed25519_sign	36.8 $\mu$ s/iter	51.9 $\mu$ s/iter
Ed25519_verify	96.7 $\mu$ s/iter	57.3 $\mu$ s/iter
parse(Sign1)	2.4 $\mu$ s/iter	
ser(Sign1)	1.0 $\mu$ s/iter	
ser(Sig_structure)	1.0 $\mu$ s/iter	

Table 2. Benchmarking results of our EverCDDL-based COSE signature implementation. We sign and verify a message with an 896 byte long payload using Ed25519. The benchmarks were compiled with clang 19.1.7 (-O3) and run on an Intel Xeon W-2255 CPU.

- We interoperate with pycose



# EverCDDL Applicability (IETF 123 Hackathon)

- We support most of:
  - [COSE](#) (RFC 9052)
  - [CWT](#), [EAT](#) (RFC 9711)
  - [draft-ietf-ace-edhoc-oscore-profile-08](#)
  - [draft-ietf-core-href-23](#)
  - [draft-ietf-core-observe-multicast-notifications-12](#)
  - ...
- Some manual rewrites due to currently unsupported features:
  - Syntactic transformations: ~, &
  - JC<\_, \_>, CBOR\_ONLY<\_>, other generics
  - Occurrence bounds
  - Disable floating-point
  - Unfold bounded recursion in COSE

# No support for recursive CDDL descriptions

- May cause unbounded stack consumption during validation
- COSE (RFC 9052): COSE-Recipients
  - Bounded by 3 in Appendix B
  - Unfolded by hand
  - Maybe “tail-recursive”? Hard to generalize
- EAT (RFC 9711): nested Claims-Set
  - Nested tokens
  - Submodules

# Some potential inaccuracies in existing CDDL descriptions?

- Missing cuts next to extension tables
  - COSE header\_map
  - EDHOC\_Information
  - DPE (DICE Protection Environment, Trusted Computing Group)
  - ...
- PEG interpretation
  - COSE Sig\_structure, etc.
- Some constraints can be added but are not
  - COSE header\_map entries for keys 5 and 6 cannot appear together

```
header_map = {  
    ?    1 => int / tstr,    ; algorithm identifier  
    ?    2 => [+label],      ; criticality  
    ?    3 => tstr / int,    ; content type  
    ?    4 => bstr,          ; key identifier  
    ?    ( 5 => bstr //      ; IV  
          6 => bstr ),       ; Partial IV  
    * label => values  
}
```

# Missing CDDL cuts in the wild

- COSE header\_map

```
header_map = {  
    ?      1 => int / tstr,    ; algorithm identifier  
    ?      2 => [+label],     ; criticality  
    ?      3 => tstr / int,    ; content type  
    ?      4 => bstr,          ; key identifier  
    ?      ( 5 => bstr //      ; IV  
             6 => bstr ),      ; Partial IV  
    * label => values  
}
```

- Also: EDHOC\_Information, etc.

# Missing CDDL cuts in the wild

- COSE header\_map

**These should be cuts  
: or ^=>**

```
header_map = {  
  ? 1 => int / tstr, ; algorithm identifier  
  ? 2 => [+label], ; criticality  
  ? 3 => tstr / int, ; content type  
  ? 4 => bstr, ; key identifier  
  ? ( 5 => bstr // ; IV  
      6 => bstr ), ; Partial IV  
  * label => values  
}
```

- Also: EDHOC\_Information, etc.

# CDDL is PEG

- Occurrence operators ?, \* should be interpreted in a greedy way without backtracking
- COSE Sig\_structure etc.

```
empty_or_serialized_map = bstr .cbor header_map / bstr .size 0
```

```
Sig_structure = [  
    context : "Signature" / "Signature1",  
    body_protected : empty_or_serialized_map,  
    ? sign_protected : empty_or_serialized_map,  
    external_aad : bstr,  
    payload : bstr  
]
```

# CDDL is PEG

- Occurrence operators ?, \* should be interpreted in a greedy way without backtracking
- COSE Sig\_structure etc.

```
empty_or_serialized_map = bstr .cbor header_map / bstr .size 0
```

```
Sig_structure = [  
  context : "Signature" / "Signature1",  
  body_protected : empty_or_serialized_map,  
  ((  
    sign_protected : empty_or_serialized_map,  
    external_aad : bstr,  
    payload : bstr  
  ) // (  
    external_aad : bstr,  
    payload : bstr  
  ))  
]
```

# EverCBOR, EverCDDL Takeaways

- Interoperable, reasonable implementations with strong formal guarantees
  - No need to hand-write parsers/serializers for CDDL-defined formats anymore
- Improved understanding of the CBOR deterministic encoding and of CDDL
- Clarification may be needed in CDDL descriptions in some existing standards
  - For use with EverCDDL and other automated code generators
- <https://github.com/project-everest/everparse>
- Tahina Ramananandro, [taramana@microsoft.com](mailto:taramana@microsoft.com)