Vest: Verified, Secure, High-Performance Parsing and Serialization for Rust

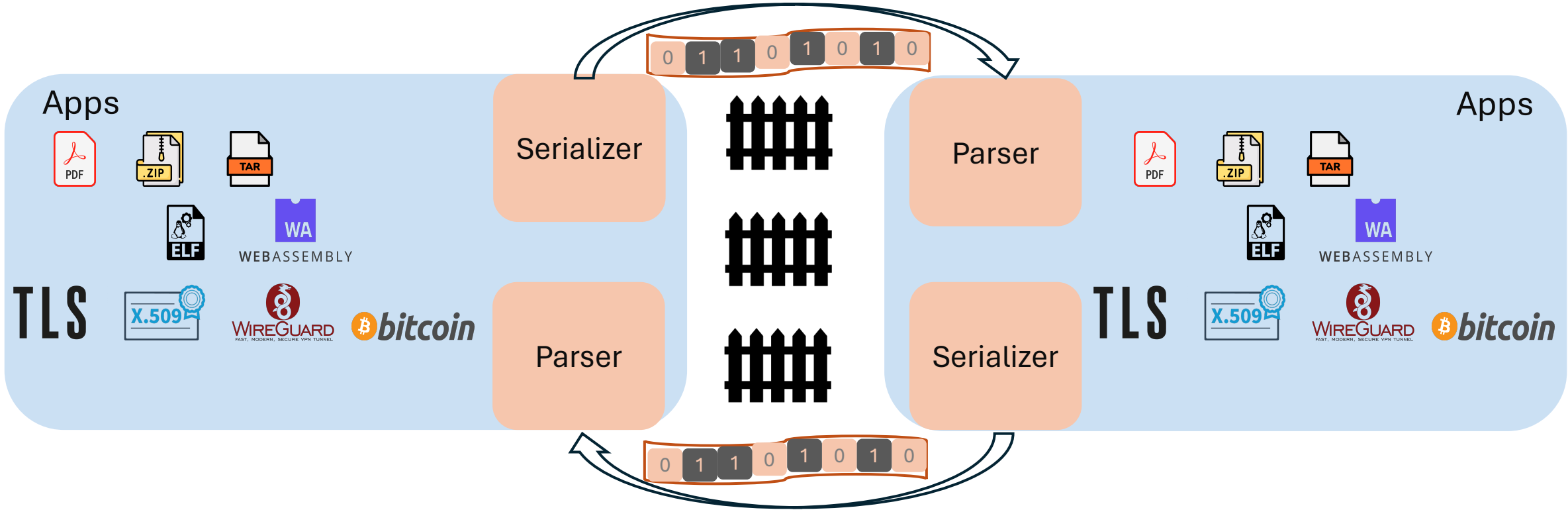
Yi Cai[1], Pratap Singh[2], Zhengyao Lin[2],
Jay Bosamiya[3], Joshua Gancher[4],
Milijana Surbatovich[1], and Bryan Parno[2]

[1]University of Maryland [2]Carnegie Mellon University

[3]Microsoft Research [4]Northeastern University

Binary Format is EVERYWHERE

Binary Format is EVERYWHERE



Secure Parsing is HARD

2019 CWE Top 25 Most Dangerous Software Errors

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61

2020 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47

2021 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score
[1]	CWE-787	Out-of-bounds Write	65.93
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84
[3]	CWE-125	Out-of-bounds Read	24.9
[4]	CWE-20	Improper Input Validation	20.47

2022 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score	Count (CVEs)
1	CWE-787	Out-of-bounds Write	64.20	62
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7
4	CWE-20	Improper Input Validation	20.63	20

Secure Parsing is HARD

Secure Parsing is HARD

Parser malleability attacks (Ramananandro, USENIX Sec'19)

Format confusion attacks (Wallez, CCS'23)

Secure Parsing is HARD

Vest: A **Verified**, **Secure**, **Performant**
Parser and Serializer **Generator** for Rust

Zero-copy parsers
In-place serializers

Vest: A **Verified**, **Secure**, **Performant**
Parser and Serializer **Generator** for Rust

Zero-copy parsers
In-place serializers

Vest: A **Verified**, **Secure**, **Performant**
Parser and Serializer **Generator** for Rust

Soundness
Completeness
Non-malleability
Non-ambiguity
No integer overflows
No panics
No infinite loops
Memory safety
...

Machine-checked proofs

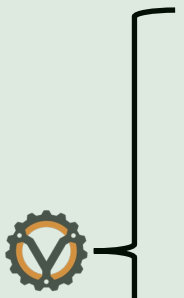
Zero-copy parsers
In-place serializers

Vest: A **Verified**, **Secure**, **Performant** Parser and Serializer **Generator** for Rust

Machine-checked proofs

Zero-copy parsers
In-place serializers

Vest: A **Verified**, **Secure**, **Performant** Parser and Serializer **Generator** for Rust

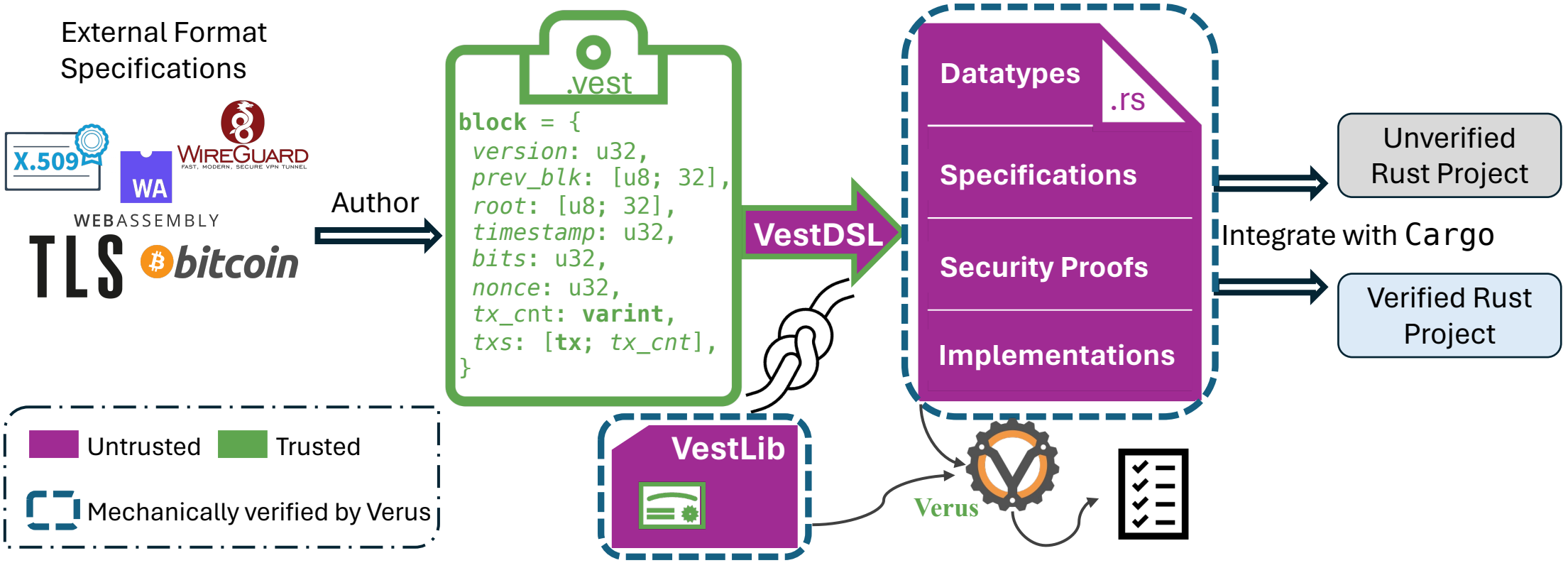


- Soundness
- Completeness
- Non-malleability
- Non-ambiguity
- No integer overflows
- No panics
- No infinite loops

- Memory safety
- ...

No expertise in formal methods required

Vest Workflow



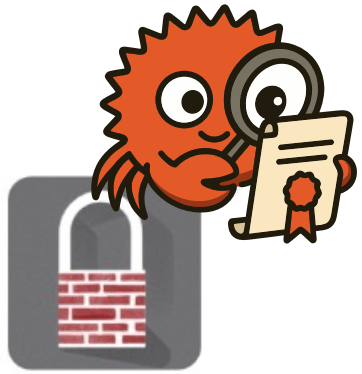
Verified Projects Using Vest

secure-foundations
/owl

Compositional Verification of Security Protocols

secure-foundations/
verdict

Formally Verified X.509 Certificate Validation



P. Singh, J. Gancher, and B. Parno. *OwlC: Compiling security protocols to verified, secure, high-performance libraries*. In Proceedings of the **USENIX Security Symposium**, Aug. 2025.

Z. Lin, M. McLoughlin, P. Singh, R. Brennan-Jones, P. Hitchcox, J. Gancher, and B. Parno. *Towards practical, end-to-end formally verified x.509 certificate validators with Verdict*. In Proceedings of the **USENIX Security Symposium**, Aug. 2025.

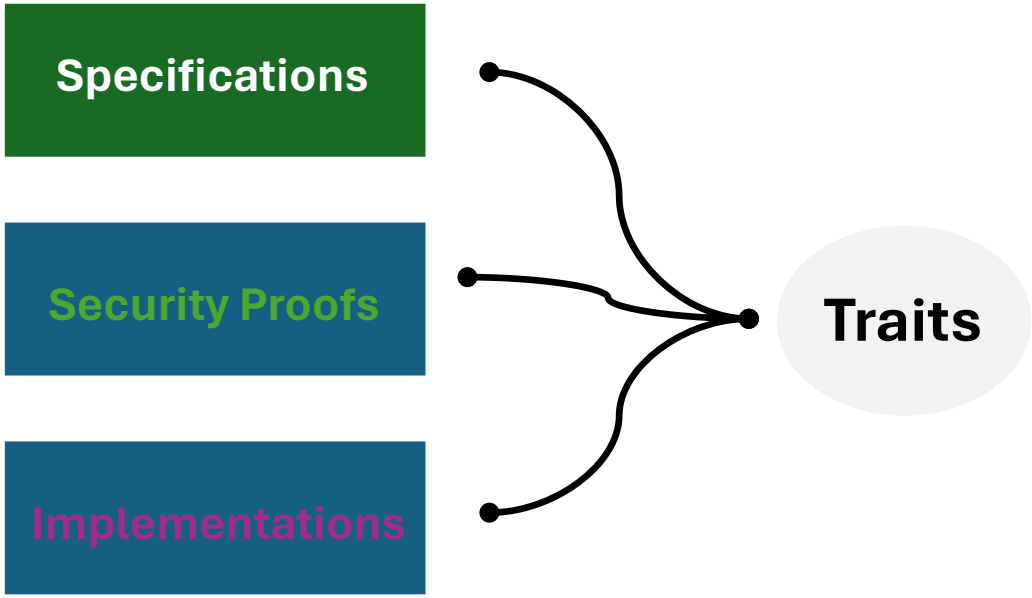
: A **Fully-compositional** ***Combinator*** **Library**

- `U8, U16Le, U24Le, …`
- `bytes::Variable(n), bytes::Fixed::<N>, …`
- `Pair(A, |a| B(a)) Choice(A, B) RepeatN(Inner, n), …`

10 Primitive Combinators +
9 Higher-order Combinators

Non-malleability
Non-ambiguity

Zero-copy **Parser**
In-place **Serializer**



Composing Parser and Serializer **Specifications**

```
msg = {
  tag: u8,
  len: u16,
  val: [u8; len] >>=
    choose (tag) {
      1 ⇒ msg1,
      2 ⇒ msg2,
  },
}
```

```
Pair(
  Pair(U8, U16),
  |(tag, len)| AndThen(
    Bytes(len),
    Choice(
      Cond(tag == 1, Msg1),
      Cond(tag == 2, Msg2)
    )
  )
)
```

```
.spec_parse(…) OR
.spec_serialize(…)
```

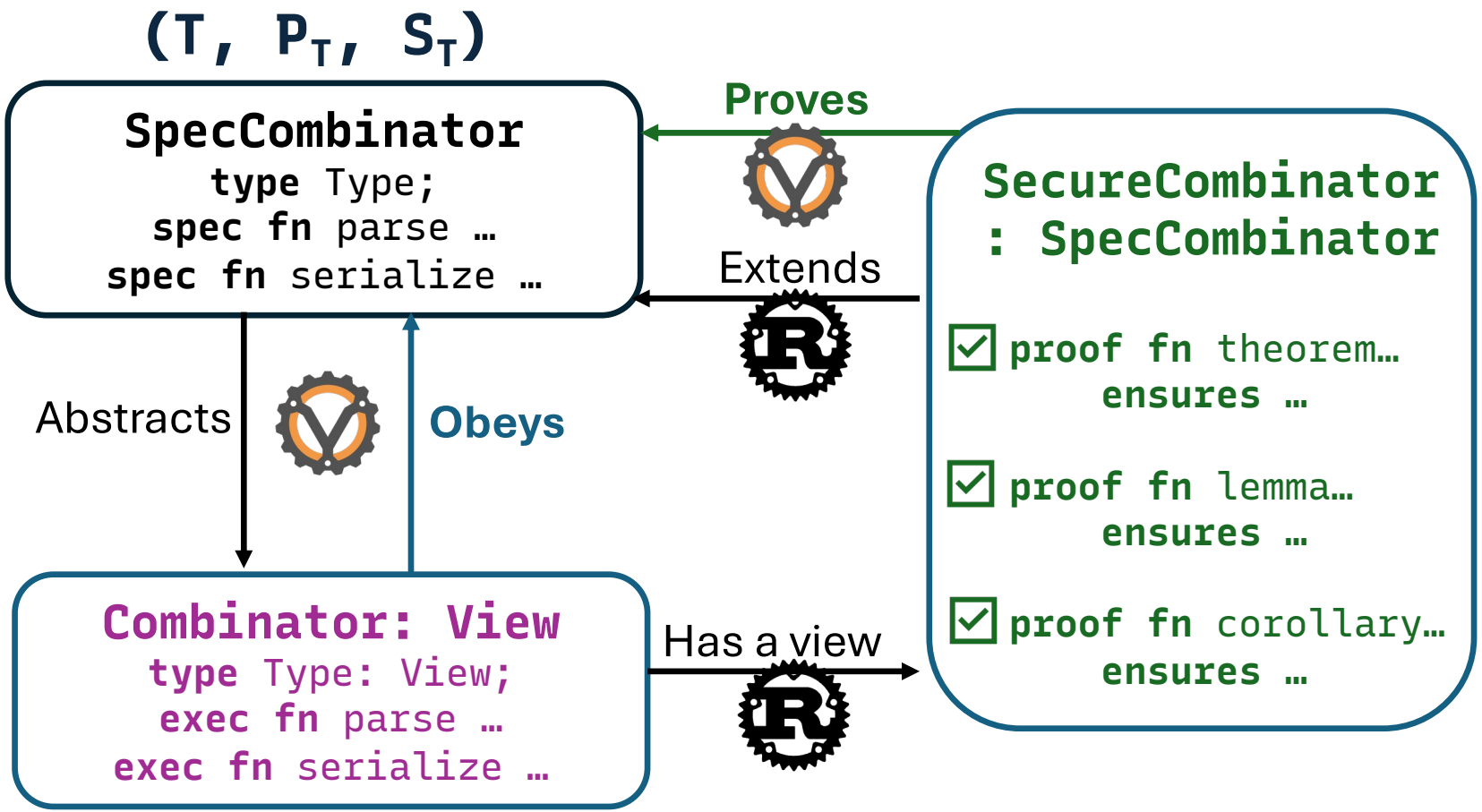
Composing Parser and Serializer **Security Proofs**

```
Pair(
  Pair(U8, U16),
  |(tag, len)| AndThen(
    Bytes(len),
    Choice(
      Cond(tag == 1, Msg1),
      Cond(tag == 2, Msg2)
    )
  )
)
.theorem_parse_serialize_roundtrip()
.theorem_serialize_parse_roundtrip()
```

Composing the **Implementation** & **Verification**

```
exec fn parse_msg(i: &'a [u8]) -> (r: ParseResult<Msg, &'a [u8])>)
    ensures
        parse_correct(i, r, spec_parse_msg)
{
    Pair(
      Pair(U8, U16),
      |(tag, len)| AndThen(
        Bytes(len),
        Choice(
          Cond(tag == 1, Msg1),
          Cond(tag == 2, Msg2)
        )
      )
    )
    .parse(i)
}
```

VestLib's Trait-based Design



VestLib's Trait-based Design

```
msg = {
  tag: u8,
  len: u16,
  val: [u8; len] >>=
    choose (tag) {
      1 ⇒ msg1,
      2 ⇒ msg2,
  },
}
```

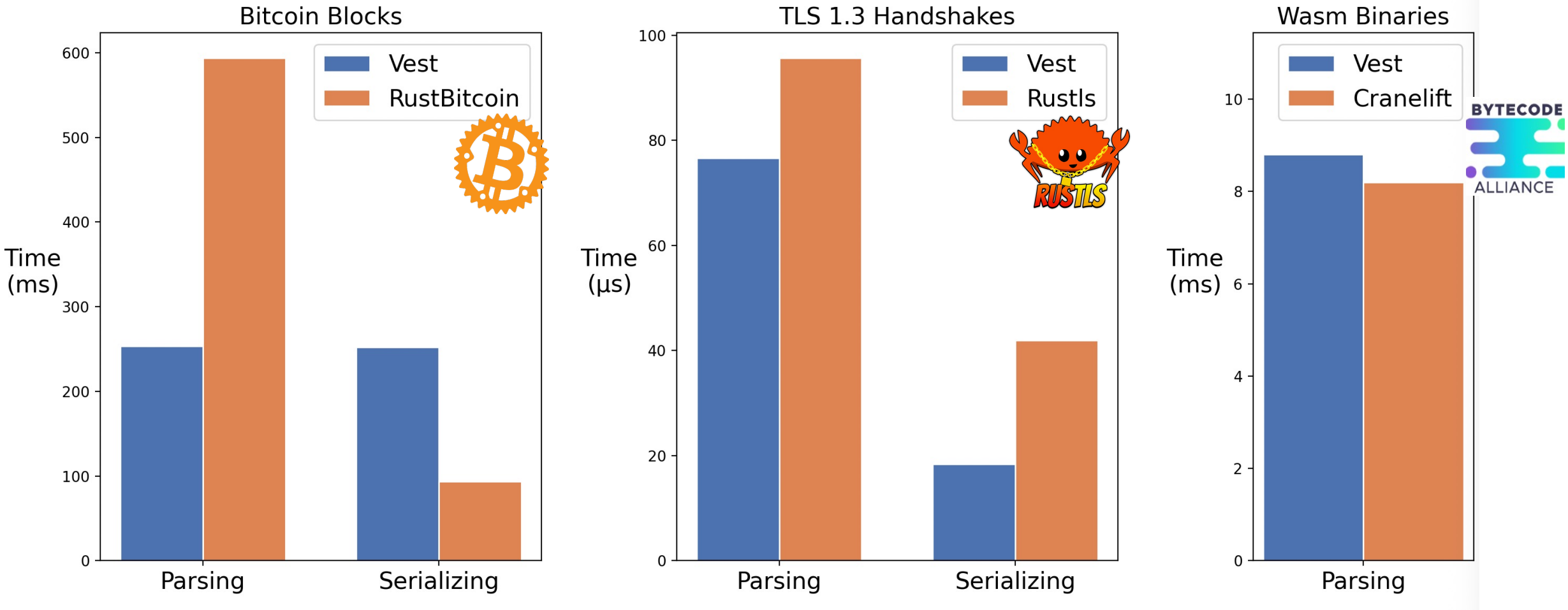
```
Pair(
  Pair(U8, U16),
  |(tag, len)| AndThen(
    Bytes(len),
    Choice(
      Cond(tag == 1, Msg1),
      Cond(tag == 2, Msg2)
    )
  )
)
```

```
.parse(…) OR
.serialize(…) OR
@.spec_parse(…) OR
@.spec_serialize(…) OR
@.theorem_serialize_parse_roundtrip() OR
@.theorem_parse_serialize_roundtrip() OR
@.lemma_parse_non_malleable()
…
```

~~Separate combinators for parsing and serialization~~
~~Individual functions for properties and proofs~~
~~Semi-compositional~~
~~HOF: LOTS of quantifiers~~

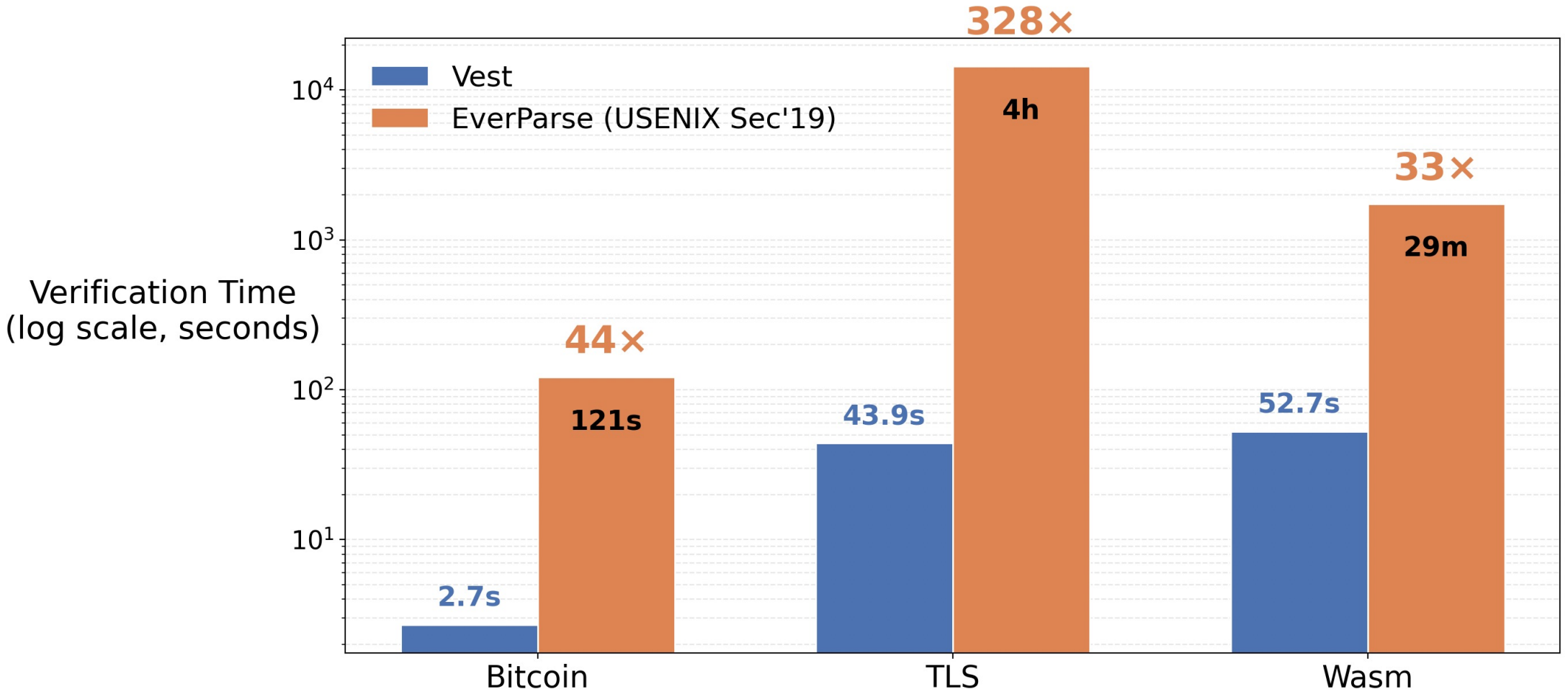
Single combinator for parsing and serialization
Associated properties and proofs
Fully-compositional
"Defunctionalized": Minimized quantifiers

Vest Execution Performance



Execution Time (lower is better)
Vest vs Unverified Industrial Baselines

Vest Verification Performance



Verification Time on Generated Code (lower is better)

We are actively working on…

- Recursive formats
- Bit-precise formats
- Pointer-rich formats
- Parsing actions
- Error resistant parsers
- Non-linear parsers
- …

Machine-checked proofs

Zero-copy parsers
In-place serializers

Vest: A **Verified**, **Secure**, **Performant** Parser and Serializer **Generator** for Rust

- Soundness
- Completeness
- Non-malleability
- Non-ambiguity
- No integer overflows
- No panics
- No infinite loops
- Memory safety
- ...

No expertise in formal methods required

Email: yicai@umd.edu

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED