Towards formally secure compilation of verified F⋆ programs against unverified ML contexts

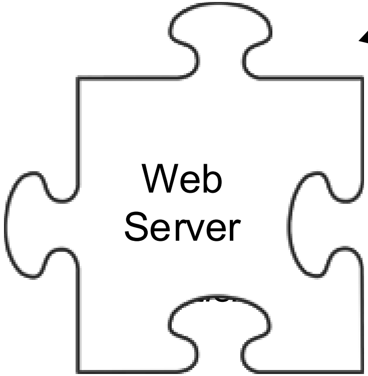
Cezar-Constantin **Andrici**, Danel **Ahman,** Cătălin **Hriţcu,**

Guido **Martínez,** Abigail **Pribisova,** Exequiel **Rivas,** Théo **Winterhalter**

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY
UNIVERSITY OF TARTU
Microsoft
TALTECH
Inria

Proof-oriented language **F⋆ offers strong guarantees**

We annotate the code with
refinement types and
pre- and post-conditions



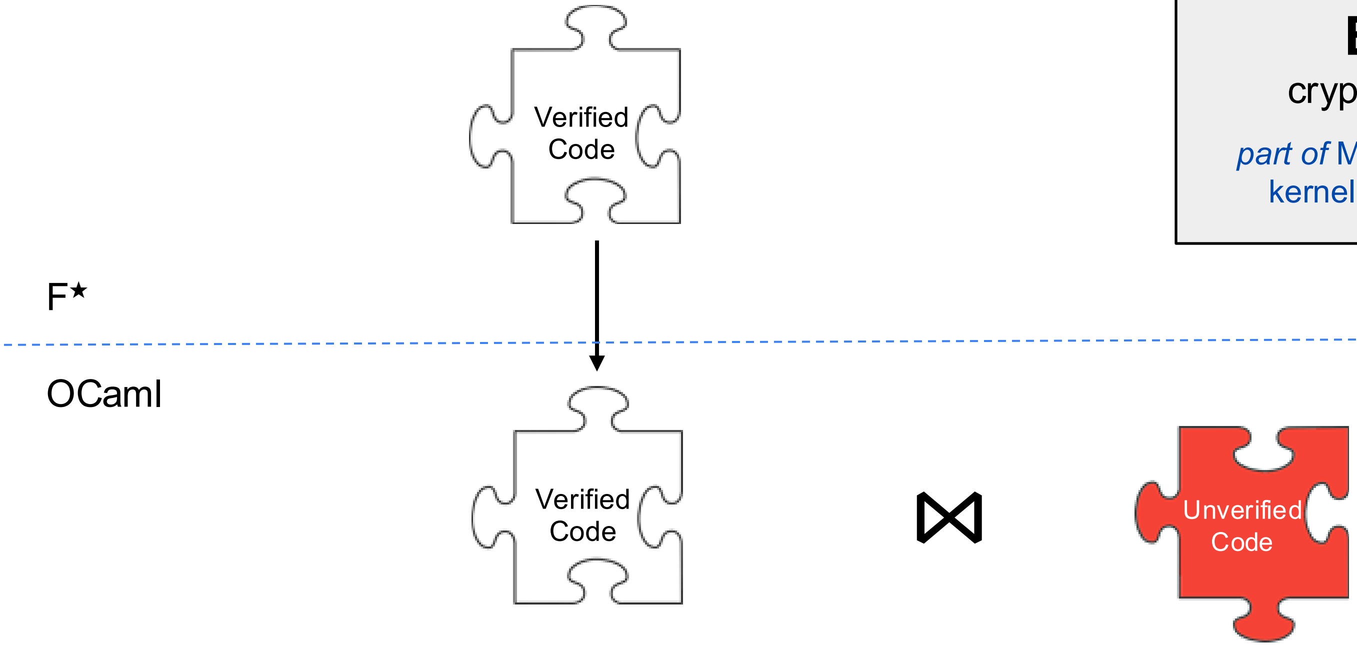
The F⋆ type checker verifies
if the code satisfies the annotations.

Specification
“responds to every request”

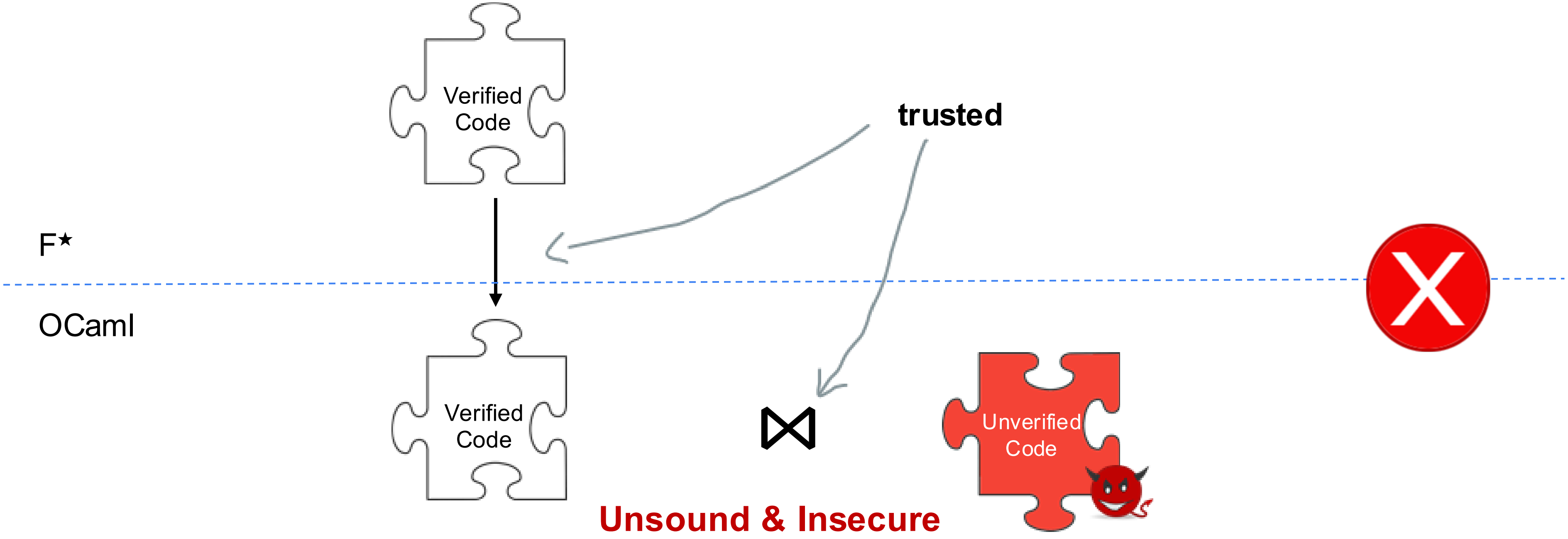
Written in F⋆, extracted to other languages

EverCrypt
cryptographic provider

part of Mozilla Firefox, the Linux kernel, the Wireguard VPN.

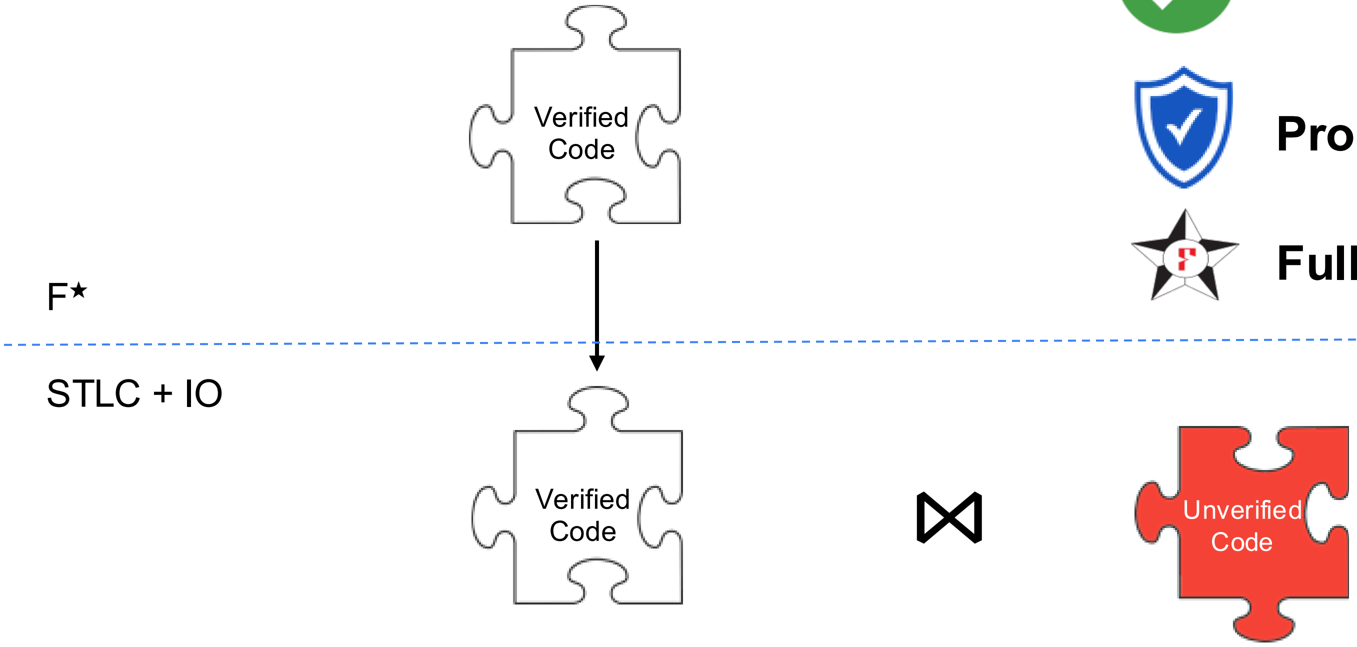


Mixing verified code with unverified code can be **problematic**

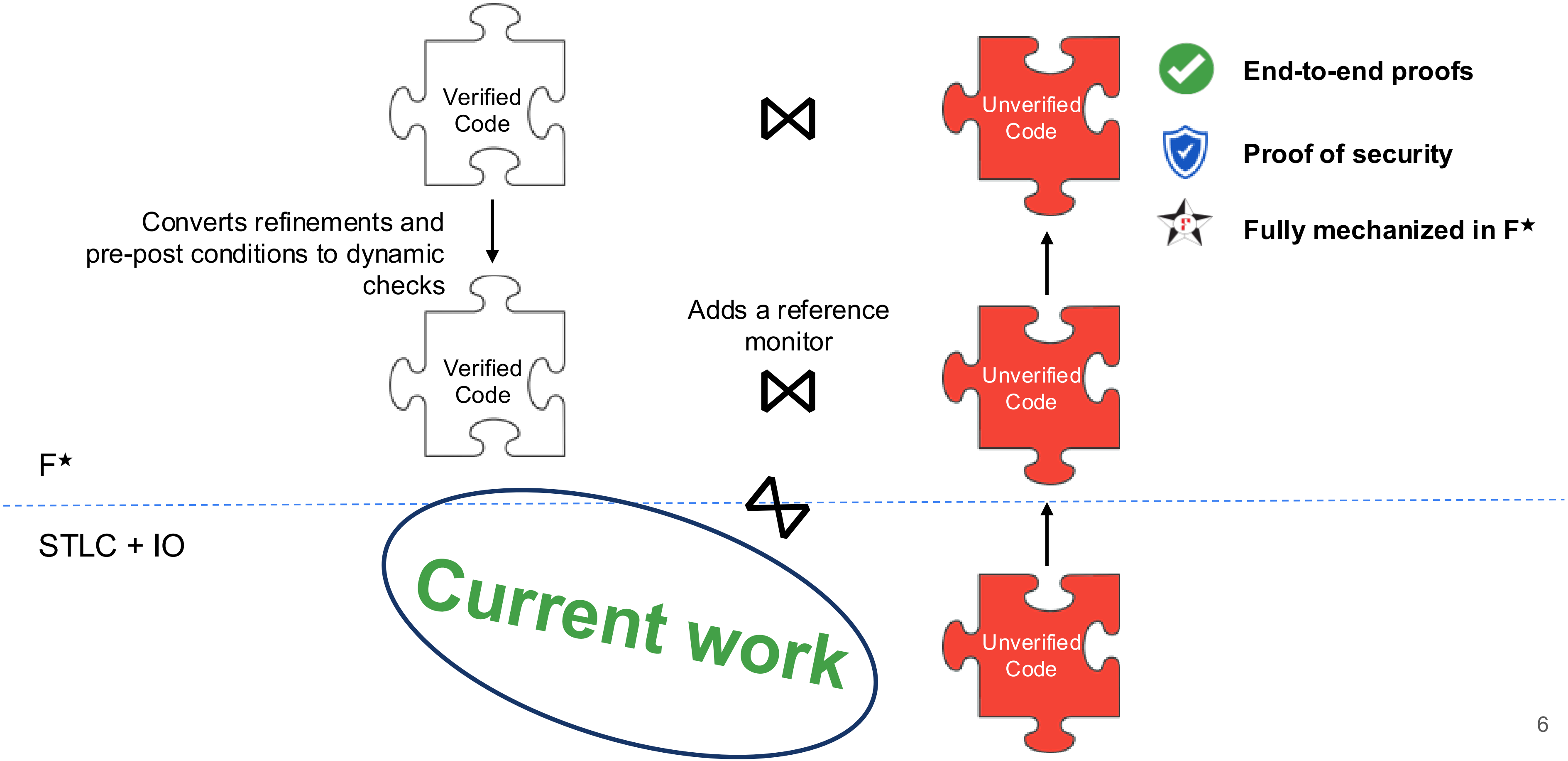


Towards secure compilation of **terminating higher-order IO programs**

- **End-to-end proofs**
- **Proof of security**
- **Fully mechanized in F⋆**



SCIO⋆: a verified secure compilation framework for verified IO programs (Andrici et al. POPL'24)



Verifying extraction end-to-end is challenging

```
let prog lib : io unit =
  let msg = read () in
  let res = lib msg in
  write res
```

Trace-producing semantics:

```
[EvRead msg; ........................  ; EvWrite res]
```

```
let comp_prog : exp =
  ELambda (
    ELet ERead (
    ELet (EApp (EVar 1) (EVar 0)) (
    EWrite (EVar 0))))
```

Shallow embedding

Compilation

One needs a meta program

If compilation uses quotation, then we have to verify it to have an end-to-end proof (requires meta theory of F⋆).

Deep embedding

We propose **Relational Quotation**

Relational quotation involves a special *typing relation* and a *meta program*.

We define a **typing relation** for the subset of F⋆ we want to compile:

```
type typing : Γ:typ_env → a:Type → (eval_env Γ → a) → Type =
| Qfalse  : Γ:typ_env → typing Γ bool (λ _  → false)
| QVar0   : Γ:typ_env → a:Type →
            typing (extend a Γ) a (λ σ → hd σ)
| QVarS   : ...
| QLambda : Γ:typ_env → a:Type → b:Type →
            body:(eval_env (extend a Γ) → b) →
            typing (extend a Γ) b body →
            typing Γ (a → b) (λ σ x → body (push σ x))
```

To support the **io** monad, we define two mutually recursive relations following the typing rules of *fine-grain call-by-value (*P.B. Levy et al. 2003).

The typing derivation captures the program's quotation

Shallow embedding

```
let prog lib : io unit =
  let msg = read () in
  let res = lib msg in
  write res
```

Typing derivation

```
let tyj_prog : typing empty ((string → io string) → io unit) prog =
  QLambda (
    QLet QRead (
    QLet (QApp QVar1 QVar0) (
    QWrite QVar0)))
```

Compiler model

Meta program

Shallow embedding

```
let prog lib : io unit =
  let msg = read () in
  let res = lib msg in
  write res
```

Typing derivation

```
let tyj_prog : typing empty _ prog =
  QLambda (
    QLet QRead (
    QLet (QApp QVar1 QVar0) (
    QWrite QVar0)))
```

Compilation

Deep embedding

```
let comp_prog : exp =
  ELambda (
    ELet ERead (
    ELet (EApp (EVar 1) (EVar 0)) (
    EWrite (EVar 0))))
```

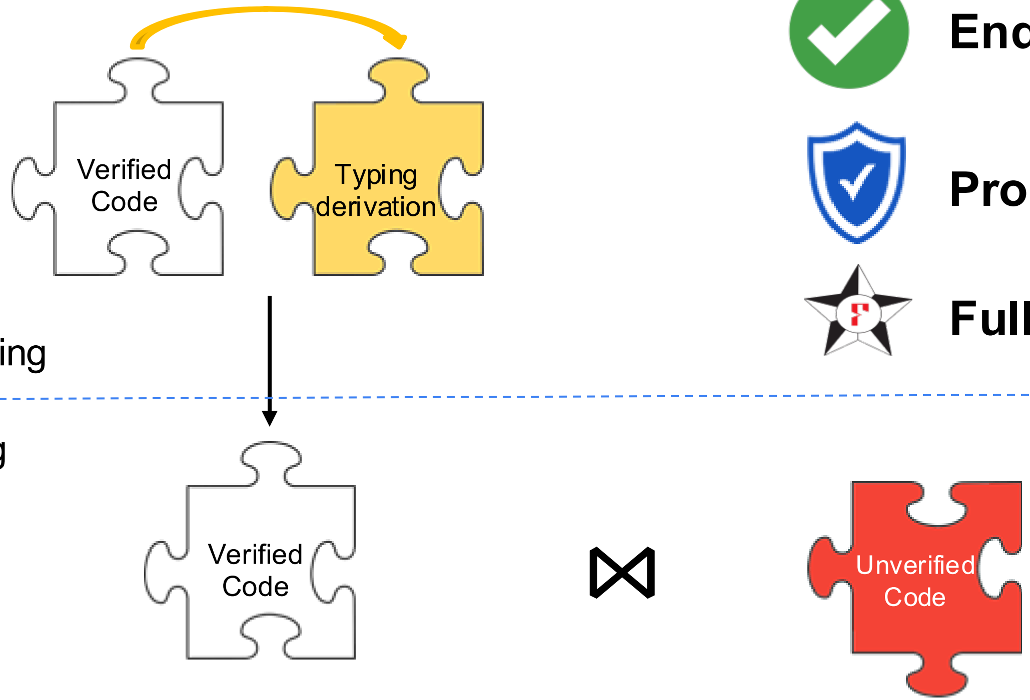
End-to-end proof

The compiler satisfies

Robust Relational Hyperproperty Preservation (RrHP)

- Strongest criterion of Abate et al. (CSF’19). **Stronger than full abstraction.**
- **Compilation preserves**:
  - Observational equivalence
  - Noninterference
  - Trace properties
- Proof using a cross language logical relation:
  - Asymmetric relation: relates shallow to deep embeddings
  - Proof done recursively on the typing derivation
- No need for the meta theory of F⋆.

Towards secure compilation of **terminating higher-order IO programs**



Shallow Embedding

Deep Embedding
(STLC+IO)

End-to-end proofs

Proof of security

Fully mechanized in F★

Cezar Andrici, MPI-SP: cezar.andrici@mpi-sp.org