Formally Verified Cryptographic Web Applications in WebAssembly

Jonathan Protzenko Benjamin Beurdouche Denis Merigoux
Karthikeyan Bhargavan

Microsoft Research & Inria (Prosecco)

May 6th 2019

Applications have to be deployed on a number of different platforms and architectures: desktop (Windows, Mac, Linux), website and smartphone application (Android and iOS). A popular solution is to write a Web application and bundle it using frameworks like Electron.

Cryptographic web applications

Applications have to be deployed on a number of different platforms and architectures: desktop (Windows, Mac, Linux), website and smartphone application (Android and iOS). A popular solution is to write a Web application and bundle it using frameworks like Electron.

Because of this, having good security means in practice ensuring good security using the technologies from the Web Stack.

Applications have to be deployed on a number of different platforms and architectures: desktop (Windows, Mac, Linux), website and smartphone application (Android and iOS). A popular solution is to write a Web application and bundle it using frameworks like Electron.

Because of this, having good security means in practice ensuring good security using the technologies from the Web Stack.

How is security managed on the Web? Standard network protection is ensured by TLS and the HTTPS protocol. But what about higher-level security guarantees?

Applications have to be deployed on a number of different platforms and architectures: desktop (Windows, Mac, Linux), website and smartphone application (Android and iOS). A popular solution is to write a Web application and bundle it using frameworks like Electron.

Because of this, having good security means in practice ensuring good security using the technologies from the Web Stack.

How is security managed on the Web? Standard network protection is ensured by TLS and the HTTPS protocol. But what about higher-level security guarantees?

Examples of cryptographic applications beyond TLS

- Storing encrypted data on servers (Lastpass).
- End-to-end encryption between devices (Whatsapp, Signal).

Applications have to be deployed on a number of different platforms and architectures: desktop (Windows, Mac, Linux), website and smartphone application (Android and iOS). A popular solution is to write a Web application and bundle it using frameworks like Electron.

Because of this, having good security means in practice ensuring good security using the technologies from the Web Stack.

How is security managed on the Web? Standard network protection is ensured by TLS and the HTTPS protocol. But what about higher-level security guarantees?

Examples of cryptographic applications beyond TLS

- Storing encrypted data on servers (Lastpass).
- End-to-end encryption between devices (Whatsapp, Signal).

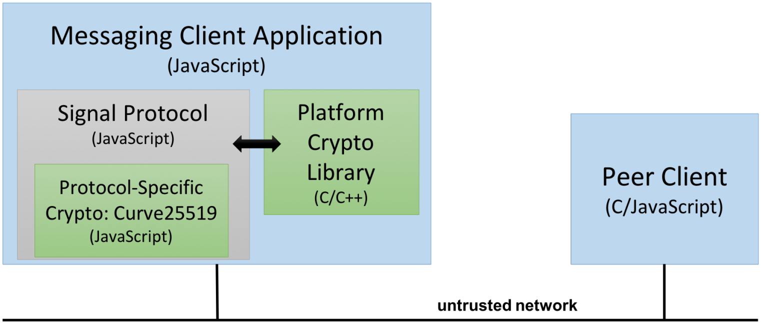
There is a need for cryptography beyond TLS, and at the application level. There are several ways to implement it.

There is a need for cryptography beyond TLS, and at the application level. There are several ways to implement it.

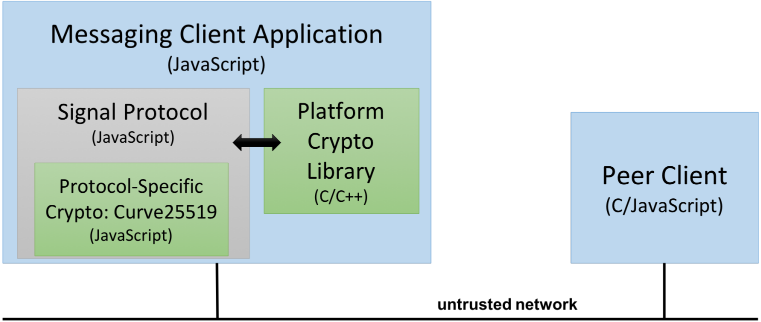
Web cryptographic solutions

- The WebCrypto API (fast, reliable, but only certain primitives)
- Custom Javascript (slow, not secure [BDLM14])
- `asm.js` (C compiled to Javascript)
- WebAssembly (new !)

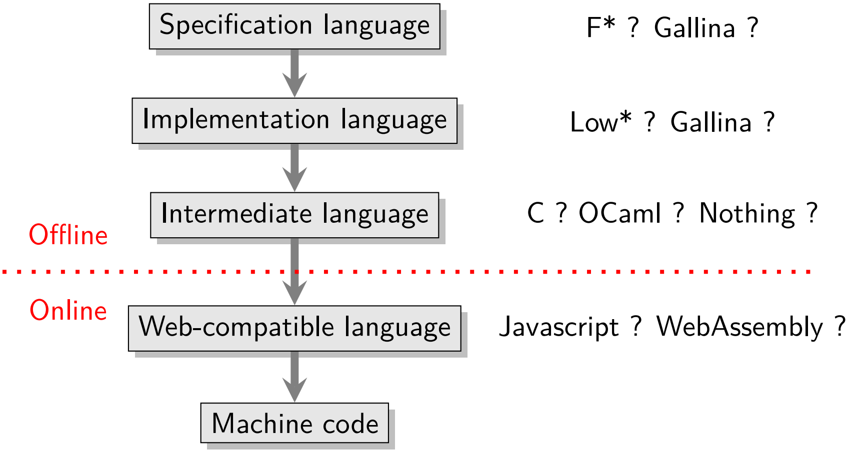
Here is how Signal implements its cryptographic protocol:



Here is how Signal implements its cryptographic protocol:



How do we get verified software inside this architecture?


Specification language
F* ? Gallina ?
Implementation language
Low* ? Gallina ?
Intermediate language
C ? OCaml ? Nothing ?
Offline
Online
Web-compatible language
Javascript ? WebAssembly ?
Machine code

WebAssembly [Haa+17] is

- a low-level intermediate representation (or a macro-assembler);
- with structured control flow;
- written as an AST;
- architecture-independent;
- typechecked before execution;
- formally specified;
- memory-management-agnostic (it gives only a flat memory buffer);
- modular with a simple import-export semantic;
- interoperable with Javascript.

```
(module
 (export "fib" (func $fib))
 (func $fib (param $n i32) (result i32)
  (if (i32.lt_s
     (get_local $n)
     (i32.const 2))
   (return (i32.const 1))
  )
  (return (i32.add
     (call $fib (i32.sub (get_local $n) (i32.const 2)))
     (call $fib (i32.sub (get_local $n) (i32.const 1)))
   ))
 )
)
```

WebAssembly is better suited to cryptographic software than Javascript (machine arithmetic, manual memory management). It is the second best choice after using the WebCrypto API.

WebAssembly is better suited to cryptographic software than Javascript (machine arithmetic, manual memory management). It is the second best choice after using the WebCrypto API.

Our verified toolchain should target it. Prosecco and Microsoft Research have already developped a toolchain from F* (Low*) to C to verify cryptographic primitives [Pro+17].

WebAssembly is better suited to cryptographic software than Javascript (machine arithmetic, manual memory management). It is the second best choice after using the WebCrypto API.

Our verified toolchain should target it. Prosecco and Microsoft Research have already developped a toolchain from F* (Low*) to C to verify cryptographic primitives [Pro+17].

Problem

- Should we translate C to WebAssembly?
- Or directly from Low* to WebAssembly?

Going via Clight

+ Reusing existing toolchains (Low* to Clight and Emscripten)
– No verified translation to WebAssembly (unless it's added to CompCert...)
– Formalization has to deal with C99 scopes and other C details
– Loss of information (e.g. immutable local variables)

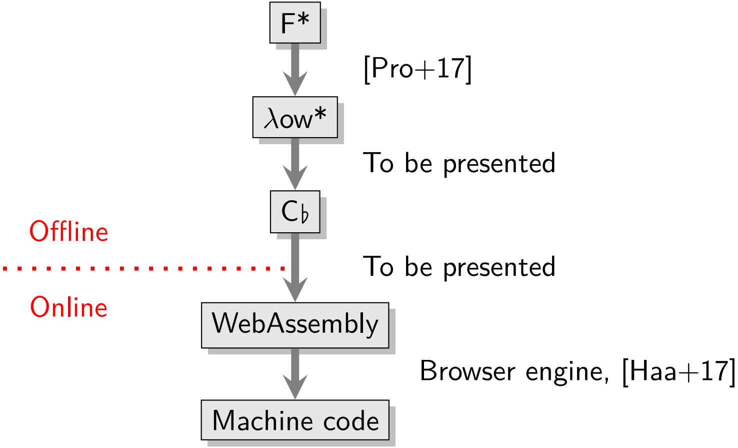
The case for a domain-specific compiler to WebAssembly

Going via Clight

- \+ Reusing existing toolchains (Low* to Clight and Emscripten)
- – No verified translation to WebAssembly (unless it's added to CompCert...)
- – Formalization has to deal with C99 scopes and other C details
- – Loss of information (e.g. immutable local variables)

Custom intermediate language C♭

- – Have to fork the existing Low* toolchain
- \+ C♭ is expression-based, no undefined behaviour
- \+ Simpler to formalize
- \+ Custom, stack-based memory management


F*
[Pro+17]
λow*
To be presented
C♭
Offline
To be presented
Online
WebAssembly
Browser engine, [Haa+17]
Machine code

F*

```
let prime = pow2 255 - 19
type elem = e:int{e >= 0 /\ e < prime}
let add e1 e2 = (e1 + e2) % prime
let mul e1 e2 = (e1 * e2) % prime
let zero: elem = 0
let one: elem = 1
```

F*

```
let prime = pow2 255 - 19
type elem = e:int{e >= 0 /\ e < prime}
let add e1 e2 = (e1 + e2) % prime
let mul e1 e2 = (e1 * e2) % prime
let zero: elem = 0
let one: elem = 1
```

Low*

```
type felem = p:uint64 p { length p = 5 }
let fadd (output a b: felem): Stack unit
  (requires (fun h0 -> live pointers h0 [output; a; b] /\
    fadd_pre h0.[a] h0.[b])
  (ensures (fun h0 h1 -> modifies only output h0 h1 /\
    h1.[output] == add h0.[a] h0.[b]))
```

$$
\begin{aligned}
\tau &::= \mathsf{int32} \mid \mathsf{int64} \mid \mathsf{unit} \mid \{\overrightarrow{f = \tau}\} \mid \mathsf{buf}\ \tau \mid \alpha \\
v &::= x \mid g \mid k : \tau \mid () \mid \{\overrightarrow{f = v}\} \\
e &::= \mathsf{readbuf}\ e_1\ e_2 \mid \mathsf{writebuf}\ e_1\ e_2\ e_3 \mid \mathsf{newbuf}\ n\ (e_1 : \tau) \\
&\mid\ \mathsf{subbuf}\ e_1\ e_2 \mid e.f \mid v \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \\
&\mid\ d\ \vec{e} \mid \mathsf{let}\ x : \tau = e_1\ \mathsf{in}\ e_2 \mid \{\overrightarrow{f = e}\} \mid e \oplus n \mid \mathsf{for}\ i \in [0; n)\ e \\
P &::= \cdot \mid \mathsf{let}\ d = \lambda \overrightarrow{y : \tau}.\ e_1 : \tau_1, P \mid \mathsf{let}\ g : \tau = e, P
\end{aligned}
$$

$$
\begin{aligned}
\tau &::= \mathsf{int32} \mid \mathsf{int64} \mid \mathsf{unit} \mid \{\overrightarrow{f = \tau}\} \mid \mathsf{buf}\ \tau \mid \alpha \\
v &::= x \mid g \mid k : \tau \mid () \mid \{\overrightarrow{f = v}\} \\
e &::= \mathsf{readbuf}\ e_1\ e_2 \mid \mathsf{writebuf}\ e_1\ e_2\ e_3 \mid \mathsf{newbuf}\ n\ (e_1 : \tau) \\
&\mid\ \mathsf{subbuf}\ e_1\ e_2 \mid e.f \mid v \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \\
&\mid\ d\ \overrightarrow{e} \mid \mathsf{let}\ x : \tau = e_1\ \mathsf{in}\ e_2 \mid \{\overrightarrow{f = e}\} \mid e \oplus n \mid \mathsf{for}\ i \in [0; n)\ e \\
P &::= \cdot \mid \mathsf{let}\ d = \lambda \overrightarrow{y : \tau}.\ e_1 : \tau_1, P \mid \mathsf{let}\ g : \tau = e, P
\end{aligned}
$$

$$
\begin{aligned}
\hat{\tau} &::= \mathsf{int32} \mid \mathsf{int64} \mid \mathsf{unit} \mid \mathsf{pointer} \\
\hat{v} &::= \ell \mid g \mid k : \hat{\tau} \mid () \\
\hat{e} &::= \mathsf{read}_n\ \hat{e} \mid \mathsf{write}_n\ \hat{e}_1\ \hat{e}_2 \mid \mathsf{new}\ \hat{e} \mid \hat{e}_1 \oplus \hat{e}_2 \mid \ell := \hat{e} \mid \hat{v} \mid \hat{e}_1;\ \hat{e}_2 \\
&\mid\ \mathsf{if}\ \hat{e}_1\ \mathsf{then}\ \hat{e}_2\ \mathsf{else}\ \hat{e}_3 : \hat{\tau} \mid \mathsf{for}\ \ell \in [0; n)\ \hat{e} \mid \hat{e}_1 \times \hat{e}_2 \mid \hat{e}_1 + \hat{e}_2 \mid d\ \overrightarrow{\hat{e}} \\
\hat{P} &::= \cdot \mid \mathsf{let}\ d = \lambda \overrightarrow{\ell : \hat{\tau}}.\overrightarrow{\ell : \hat{\tau}}, \hat{e} : \hat{\tau}, \hat{P} \mid \mathsf{let}\ g : \hat{\tau} = \hat{e}, \hat{P}
\end{aligned}
$$

λow* to C♭: desugaring structure values

$$
\begin{array}{lll}
\text{let } d = \lambda y : \tau_1 .\ e : \tau_2 & \leadsto & \text{let } d = \lambda y : \text{buf } \tau_1 .\ [\text{readbuf } y\ 0 / y] e : \tau_2 \\
\text{let } d = \lambda y : \tau_1 .\ e : \tau_2 & \leadsto & \text{let } d = \lambda y : \tau_1 .\ \lambda r : \text{buf } \tau_2 .\ \text{let } x : \tau_2 = e \text{ in writebuf } r\ 0\ x : \text{unit} \\
f\ (e : \tau) & \leadsto & \text{let } x : \text{buf } \tau = \text{newbuf } 1\ e \text{ in } f\ x \\
(f\ e) : \tau & \leadsto & \text{let } x : \text{buf } \tau = \text{newbuf } 1\ (\_ : \tau) \text{ in } f\ e\ x;\ \ \text{readbuf } x\ 0 \\
\text{let } x : \tau = e_1 \text{ in } e_2 & \leadsto & \text{let } x : \text{buf } \tau = \text{take\_addr } e_1 \text{ in } [\text{readbuf } x\ 0 / x] e_2 \\
\{\overrightarrow{f = e}\} \quad \text{(not under newbuf)} & \leadsto & \text{let } x : \text{buf } \{\overrightarrow{f = \tau}\} = \text{newbuf } 1\ \{\overrightarrow{f = e}\} \text{ in readbuf } x\ 0 \\
\text{take\_addr}(\text{readbuf } e\ n) & \leadsto & \text{subbuf } e\ n \\
\text{take\_addr}((e : \overrightarrow{f : \tau}).f) & \leadsto & \text{take\_addr}(e) \oplus \text{offset}(\overrightarrow{f : \tau}, f) \\
\text{take\_addr}(\text{let } x : \tau = e_1 \text{ in } e_2) & \leadsto & \text{let } x : \tau = e_1 \text{ in take\_addr } e_2 \\
\text{take\_addr}(\text{if } e_1 \text{ then } e_2 \text{ else } e_3) & \leadsto & \text{if } e_1 \text{ then take\_addr } e_2 \text{ else take\_addr } e_3
\end{array}
$$

$$
\begin{array}{lcll}
\text{size int32} & = & 4 & \\
\text{size unit} & = & 4 & \\
\text{size int64} & = & 8 & \\
\text{size buf } \tau & = & 4 & \\
\text{size } \overrightarrow{f : \tau} & = & \text{offset } (\overrightarrow{f : \tau}, f_n) + \text{size } \tau_n & \\
\text{offset } (\overrightarrow{f : \tau}, f_0) & = & 0 & \\
\text{offset } (\overrightarrow{f : \tau}, f_{i+1}) & = & \text{align}(\text{offset } (\overrightarrow{f : \tau}, f_i) + \text{size } \tau_i, \text{alignment } \tau_{i+1}) & \\
\text{alignment}(\overrightarrow{f : \tau}) & = & 8 & \\
\text{alignment}(\tau) & = & \text{size } \tau & \text{otherwise} \\
\text{align}(k, n) & = & k & \text{if } k \bmod n = 0 \\
\text{align}(k, n) & = & k + n - (k \bmod n) & \text{otherwise}
\end{array}
$$

$$\textsc{Let}\ \frac{\ell \text{ fresh} \qquad \begin{array}{c} G; V \vdash e_1 : \tau_1 \Rightarrow \hat{e}_1 : \hat{\tau}_1 \dashv V' \\ G; (x \mapsto \ell, \hat{\tau}_1) \cdot V' \vdash e_2 : \tau_2 \Rightarrow \hat{e}_2 : \hat{\tau}_2 \dashv V'' \end{array}}{G; V \vdash \text{let } x : \tau_1 = e_1 \text{ in } e_2 : \tau_2 \Rightarrow \ell := \hat{e}_1; \hat{e}_2 : \hat{\tau}_2 \dashv V''}$$

$$\textsc{FunDecl}\ \frac{G; \overrightarrow{y \mapsto \ell, \hat{\tau}} \vdash e_1 : \tau_1 \Rightarrow \hat{e}_1 : \hat{\tau}_1 \dashv \overrightarrow{x \mapsto \ell', \hat{\tau}'} \cdot \overrightarrow{y \mapsto \ell, \hat{\tau}}}{G \vdash \text{let } d = \lambda \overrightarrow{y : \tau}.\ e_1 : \tau_1 \Rightarrow \text{let } d = \lambda \overrightarrow{\ell : \hat{\tau}}.\overrightarrow{\ell' : \hat{\tau}'}, \hat{e}_1 : \hat{\tau}_1}$$

$$\textsc{Var}\ \frac{V(x) = \ell, \tau}{G; V \vdash x \Rightarrow \ell : \tau \dashv V}$$

$$\textsc{BufWrite}\ \frac{G; V \vdash \text{writeB } (e_1 + e_2 \times \text{size } \tau_1)\ e_3 \Rightarrow \hat{e} \dashv V'}{G; V \vdash \text{writebuf } (e_1 : \tau_1)\ e_2\ e_3 \Rightarrow \hat{e} : \text{unit} \dashv V'}$$

$$\textsc{WriteInt32}\quad \frac{G; V \vdash e_1 \Rrightarrow \hat{e}_1 \dashv V' \qquad G; V' \vdash e_2 \Rrightarrow \hat{e}_2 \dashv V''}{G; V \vdash \mathsf{writeB}\ e_1\ (e_2 : \mathsf{int32}) \Rrightarrow \mathsf{write}_4\ \hat{e}_1\ \hat{e}_2 \dashv V''}$$

$$\textsc{WriteLiteral}\quad \frac{G; V_i \vdash \mathsf{writeB}\ (e + \mathsf{offset}\ (\overrightarrow{f : \tau}, f_i))\ e_i \Rrightarrow \hat{e}_i \dashv V_{i+1}}{G; V_0 \vdash \mathsf{writeB}\ e\ (\{\overrightarrow{f = e : \tau}\}) \Rrightarrow \hat{e}_0; \ldots; \hat{e}_{n-1} \dashv V_n}$$

$$\textsc{WriteDeRef}\quad \frac{\begin{array}{c}\ell\ \mathsf{fresh} \qquad V' = \ell, \mathsf{int32} \cdot V \qquad G; V \vdash v_i \Rrightarrow \hat{v}_i \dashv V \\ \mathsf{memcpy}\ v_1\ v_2\ n = \mathsf{for}\ \ell \in [0; n)\ \mathsf{write}_1\ (v_1 + \ell)\ (\mathsf{read}_1\ (v_2 + \ell)\ 1)\end{array}}{G; V \vdash \mathsf{writeB}\ v_1\ (\mathsf{readbuf}\ (v_2 : \tau_2)\ 0) \Rrightarrow \mathsf{memcpy}\ v_1\ v_2\ (\mathsf{size}\ \tau_2) \dashv V'}$$

BufNew

$$\frac{\ell, \ell' \text{ fresh} \qquad G; x \mapsto (\ell, \mathsf{int32}) \cdot y \mapsto (\ell', \mathsf{int32}) \cdot V \vdash \mathsf{writeB}\ (x + \mathsf{size}\ \tau \times y)\ v_1 \Rightarrow \hat{e} \dashv V'}{G; V \vdash \mathsf{newbuf}\ n\ (v : \tau) \Rightarrow \ell := \mathsf{new}\ (n \times \mathsf{size}\ \tau);\ \mathsf{for}\ \ell' \in [0; n)\ \hat{e};\ \ell \dashv V'}$$

We adopt a stack-based memory allocation scheme with a watermark at address 0.

$$
\begin{array}{lcl}
\mathsf{get\_stack} & = & \mathsf{func}\ [\,] \to \mathsf{i32}\ \mathsf{local}\ [\,] \\
 & & \mathsf{i32.const}\ 0;\ \ \mathsf{i32.load} \\
\mathsf{set\_stack} & = & \mathsf{func}\ \mathsf{i32} \to [\,]\ \mathsf{local}\ \overrightarrow{\ell : \mathsf{i32}} \\
 & & \mathsf{i32.const}\ 0;\ \ \mathsf{get\_local}\ \ell;\ \ \mathsf{i32.store} \\
\mathsf{grow\_stack} & = & \mathsf{func}\ \mathsf{i32} \to \mathsf{i32}\ \mathsf{local}\ \overrightarrow{\ell : \mathsf{i32}} \\
 & & \mathsf{call}\ \mathsf{get\_stack};\ \ \mathsf{get\_local}\ \ell;\ \ \mathsf{i32.op+}; \\
 & & \mathsf{call}\ \mathsf{set\_stack};\ \ \mathsf{call}\ \mathsf{get\_stack}
\end{array}
$$

Write32

$$\frac{\hat{e}_1 \Rrightarrow \overrightarrow{i_1} \qquad \hat{e}_2 \Rrightarrow \overrightarrow{i_2}}{\text{write}_4\ \hat{e}_1\ \hat{e}_2 \Rrightarrow \overrightarrow{i_1};\ \overrightarrow{i_2};\ \text{i32.store};\ \text{i32.const}\ 0}$$

New

$$\frac{\hat{e} \Rrightarrow \overrightarrow{i}}{\text{new}\ \hat{e} \Rrightarrow \overrightarrow{i};\ \text{call grow\_stack}}$$

For

$$\frac{\hat{e} \Rrightarrow \overrightarrow{i}}{\begin{array}{l}\text{for}\ \ell \in [0; n)\ \hat{e} \Rrightarrow \\ \text{loop}(\ \overrightarrow{i}\ ;\text{drop}; \\ \quad \text{get\_local}\ \ell; \text{i32.const}\ 1; \text{i32.op+}; \text{tee\_local}\ \ell; \\ \quad \text{i32.const}\ n; \text{i32.op} =; \text{br\_if}); \text{i32.const}\ 0\end{array}}$$

Func

$$\frac{\hat{e} \Rrightarrow \overrightarrow{i} \qquad \hat{\tau}_i \Rrightarrow t_i}{\begin{array}{l}\text{let } d = \lambda\overrightarrow{\ell_1 : \hat{\tau}_1}.\overrightarrow{\ell_2 : \hat{\tau}_2}, \hat{e} : \hat{\tau} \Rrightarrow \\ d = \text{func } \overrightarrow{t_1} \rightarrow t \text{ local } \overrightarrow{\ell_1 : t_1 \cdot \ell_2 : t_2} \cdot \ell : t. \\ \qquad \text{call get\_stack; } \overrightarrow{i}\text{ ; store\_local } \ell\text{ ; call set\_stack; get\_local } \ell\end{array}}$$

Example: compiled `fadd` function

$fadd$ = func [int32; int32; int32] → []
local [ℓ_0, ℓ_1, ℓ_2 : int32; ℓ_3 : int32; ℓ : int32].
call get_stack; loop(
// Push dst + 8*i on the stack
get_local ℓ_0; get_local ℓ_3; i32.const 8; i32.binop$*$; i32.binop+
// Load a + 8*i on the stack
get_local ℓ_1; get_local ℓ_3; i32.const 8; i32.binop$*$; i32.binop+
i64.load
// Load b + 8*i on the stack (elided, same as above)
// Add a.[i] and b.[i], store into dst.[i]
i64.binop+; i64.store
// Per the rules, return unit
i32.const 0; drop
// Increment i; break if i == 5
get_local ℓ_3; i32.const 1; i32.binop+; tee_local ℓ_3
i32.const 5; i32.op =; br_if
); i32.const 0
store_local ℓ ; call set_stack; get_local ℓ

The compiler, KreMLin, is 11,000 LOC. The translation is implemented following this formalization, and is designed to be auditable.
We left as future work the task of replicating and adapting the translation correctness of [Pro+17] from λow* to Clight:

Lemma

Let P be a λow^* program and e be a λow* entry point expression, and assume that they compile: $\Downarrow(P) = \hat{P}$ for some C* program $\hat{P}$ and $\downarrow(e) = \overrightarrow{s};\hat{e}$ for some C* list of statements $\overrightarrow{s}$ and expression $\hat{e}$.
Let V be a mapping of local variables containing the initial values of secrets. Then, the C* program $\hat{P}$ terminates with trace ℓ and return value v, i.e.,
$\hat{P} \vdash ([], V, \overrightarrow{s}; \text{return } \hat{e}) \overset{\ell,*}{\rightarrow} ([], V', \text{return } v)$ if, and only if, so does the λow^* program:
$P \vdash (\{\}, e[V]) \overset{\ell,*}{\rightarrow} (H', v)$; and similarly for divergence.

Theorem

From [Pro+17], proven for the translation from λow* to Clight: given

1. a program well-typed against a secret interface, Γ_s, i.e, $\Gamma_s, \Gamma_P; \Sigma; \Gamma \vdash (H, e) : \tau$,
2. a well-typed implementation of the Γ_s interface, $\Gamma_s; \Sigma; \cdot \vdash_\Delta P_s$, such that P_s is equivalent modulo secrets,
3. a pair (ρ_1, ρ_2) of well-typed substitutions for Γ,

then either:

1. both programs cannot reduce further, i.e. $P_s, P \vdash (H, e)[\rho_1] \not\rightarrow$ and $P_s, P \vdash (H, e)[\rho_2] \not\rightarrow$, or
2. both programs make progress with the same trace, i.e. there exists $\Sigma' \supseteq \Sigma, \Gamma' \supseteq \Gamma, H', e'$, a pair (ρ_1', ρ_2') of well-typed substitutions for Γ', and a trace ℓ such that
   i) $P_s, P \vdash (H, e)[\rho_1] \rightarrow_\ell^+ (H', e')[\rho_1']$ and $P_s, P \vdash (H, e)[\rho_2] \rightarrow_\ell^+ (H', e')[\rho_2']$, and
   ii) $\Gamma_s, \Gamma_P; \Sigma'; \Gamma' \vdash (H', e') : \tau$

Classify

$$\frac{C \vdash i : \pi}{C \vdash i : \sigma}$$

BinOpPub

$$\frac{o \text{ is constant-time}}{C \vdash t.\text{binop } o : m\ m \to m}$$

BinOpPriv

$$\frac{o \text{ is not constant-time}}{C \vdash t.\text{binop } o : \pi\ \pi \to \pi}$$

Load

$$\frac{}{C \vdash t.\text{load} : *\sigma\ \pi \to \sigma}$$

Local

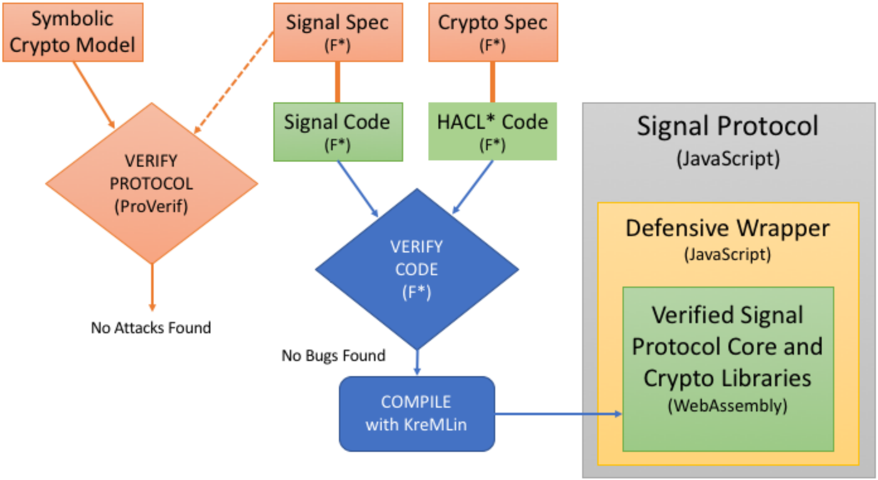
$$\frac{C(\ell) = m}{C \vdash \text{get\_local } \ell : [] \to m}$$

Cond

$$\frac{C \vdash \overrightarrow{i_1} : \overrightarrow{m} \to \pi \qquad C \vdash \overrightarrow{i_{\{2,3\}}} : \overrightarrow{m} \to \overrightarrow{m}}{C \vdash \text{if } \overrightarrow{i_1} \text{ then } \overrightarrow{i_2} \text{ else } \overrightarrow{i_3} : \overrightarrow{m}\ \pi \to \overrightarrow{m}}$$

Primitive (blocksize, #rounds)	HACL*	libsodium	WHACL*
Curve25519 (1k)	0.83 s	0.15 s	4.05 s
Chacha20 (4kB, 100k)	1.86 s	1.74 s	6.62 s
Salsa20 (4kB, 100k)	1.55 s	2.24 s	5.52 s
Ed25519 sign (16kB, 1k)	3.01 s	0.27 s	15.6 s
Ed25519 verify (16kB, 1k)	3.07 s	0.24 s	15.6 s
Poly1305_32 (16kB, 10k)	0.27 s	0.19 s	_
Poly1305_64 (16kB, 10k)	1.93 s	0.19 s	11.5 s
SHA2_256 (16kB, 10k)	1.64 s	1.84 s	3.5 s
SHA2_512 (16kB, 10k)	1.16 s	1.21 s	3.2 s

Application : the Signal protocol



WebAssembly is the best available target for verified applications on the Web. Because it is small and simple, it is also a good compilation target for domain-specific languages. [Wat+19] proposes to extend it with secretness concepts.

WebAssembly is the best available target for verified applications on the Web. Because it is small and simple, it is also a good compilation target for domain-specific languages. [Wat+19] proposes to extend it with secretness concepts.

Adapting the proof from [Pro+17] is the most direct path to improve our confidence in the correctness of the translation to WebAssembly. However, the Trusted Code Base is quite big with the F* compiler and Kremlin. A WebAssembly backend for CompCert could be a significant contribution towards a certified Web-compatible toolchain.

WebAssembly is the best available target for verified applications on the Web. Because it is small and simple, it is also a good compilation target for domain-specific languages. [Wat+19] proposes to extend it with secretness concepts.

Adapting the proof from [Pro+17] is the most direct path to improve our confidence in the correctness of the translation to WebAssembly. However, the Trusted Code Base is quite big with the F* compiler and Kremlin. A WebAssembly backend for CompCert could be a significant contribution towards a certified Web-compatible toolchain.

The Signal case study demonstrate the ability of the F* and Pro/CryptoVerif ecosystem to handle the proof of non-trivial properties (including functional correctness and security) of a large codebase, that can be extracted with a high confidence to portable code.

[BDLM14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Sergio Maffeis. "Defensive JavaScript". In: *Foundations of Security Analysis and Design VII*. Springer, 2014, pp. 88–123.

[Haa+17] Andreas Haas et al. "Bringing the Web Up to Speed with WebAssembly". In: *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 2017, pp. 185–200. ISBN: 978-1-4503-4988-8.

[Pro+17] Jonathan Protzenko et al. "Verified Low-level Programming Embedded in F*". In: *Proceedings of the ACM on Programming Languages* 1.ICFP (Aug. 2017), 17:1–17:29. ISSN: 2475-1421.

[Wat+19] Conrad Watt et al. "CT-wasm: Type-driven Secure Cryptography for the Web Ecosystem". In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 77:1–77:29. ISSN: 2475-1421. DOI: 10.1145/3290390. URL: http://doi.acm.org/10.1145/3290390.