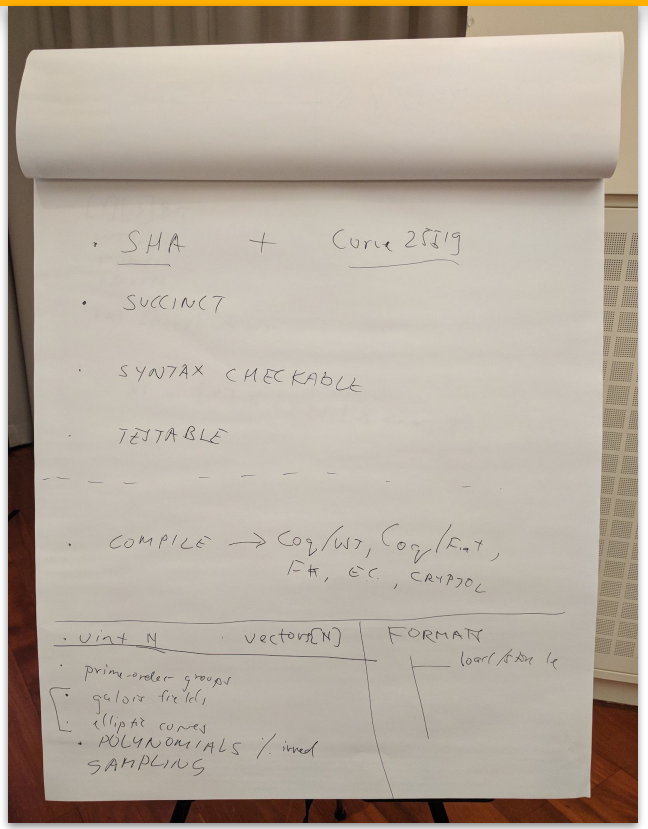
hacspec

towards verifiable crypto standards

Karthikeyan Bhargavan, **Franziskus Kiefer**, and Pierre-Yves Strub

HACS Workshop

HACS Workshop



Formal Verification

Formal verification in Crypto

Cryptol

The Language of Cryptography

Formal verification in Crypto

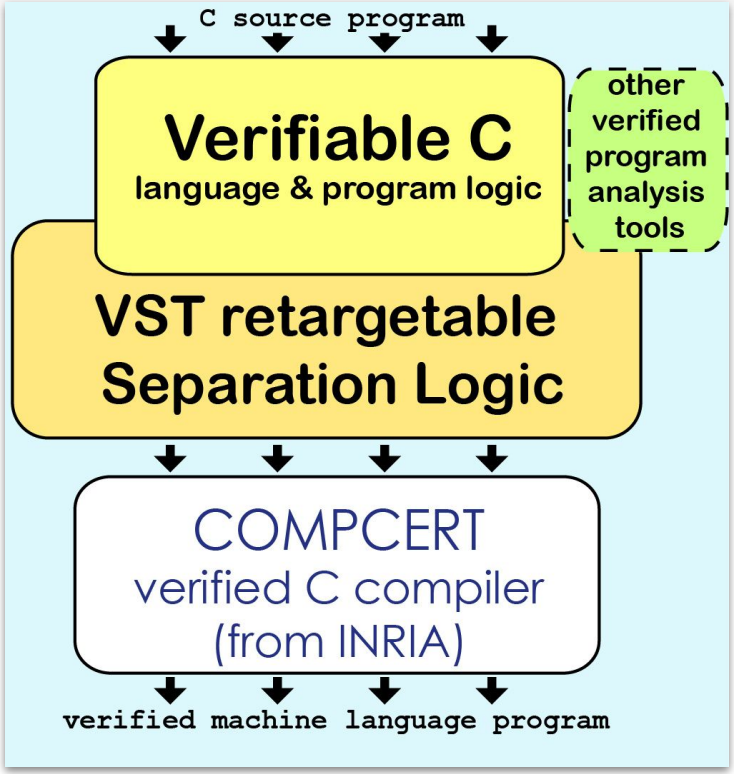
Cryptol

Proofs of correctness of the TLS Handshake and corking state machine #565

Merged alexw91 merged 116 commits into awslabs:master from GaloisInc:master on Aug 29, 2017

The Language of Cryptography

Formal verification in Crypto



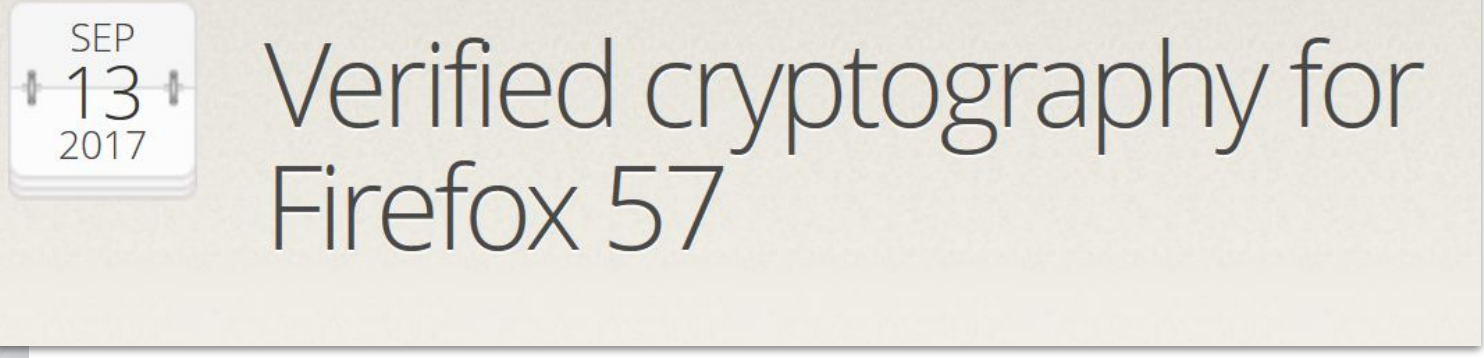
Formal verification in Crypto

Fiat-Crypto: Synthesizing Correct-by-Construction Code for Cryptographic Primitives

Formal verification in Crypto

Formal verification in Crypto

HACL*: A Verified Modern Cryptographic Library



hacspec

a new specification language for crypto primitives

Goals, Scope & Limitations

- Cryptographic Primitives
- Executable
- Easy to use
- Syntax checkable
- Compiles to several formal languages
- **Not** a new formal language

hacspec syntax

Values v ::=	`n`	*integer constants*
	`| True | False`	*boolean constants*
	`| '...' | "..."`	*string constants*
	`| (v1,...,vn)`	*tuple constant*
	`| array([v1,...,vn])`	*array constant*

hacspec syntax

Expressions e ::=	`v`	*values*
	`| x | m.x`	*local and global variables*
	`| (e1,...,en)`	*tuple construction*
	`| array([e1,...,en])`	*array construction*
	`| array.length(e)`	*array length*
	`| e[e0]`	*array access*
	`| e[e0:e1]`	*array slice*
	`| e(e1,...en)`	*function call*
	`| e1 binop e2`	*builtin binary operators*
	`| unaryop e`	*builtin unary operators*

hacspec syntax

Types `t` ::=	`int, str, bool`	*basic types*
	| `tuple_t(t1,...tn)`	*tuples*
	| `vlarray_t(t)`	*variable-length array*
	| `x`	*user-defined or builtin type*
	| `x(t1,...,tn,e1,...,em)`	*builtin type application*

hacspec syntax

Statements s ::=	`x: Type = t`	*type declaration*
	`x: t`	*variable declaration*
	`x = e`	*variable assignment*
	`x binop= e`	*augmented variable assignment*
	`(x1,..,xn) = e`	*tuple matching*
	`x[i] = e`	*array update*
	`x[i] binop= e`	*augmented array update*
	`x[i:j] = e`	*array slice update*

	`if e:`
	`  s1...sn`
	`elif e:`
	`  s1'...sn'`
	`else`
	`  s1''...sn''`
		if-elif-else conditional

	`for i in range(e):` `  s1...sn`	*for loop*
	`break`	*break from loop*
	`def x(x1:t1,...,xn:tn) → t:` `  s1 ... sn`	*function declaration*
	`return e`	*return from function*
	`from x import x1, x2,..., xn`	*module import*

hacspec speclib

- Machine integers
  - uint32_t
- Modular arithmetic
  - natmod_t(p)
- Vectors & Matrices
  - vector_t(nat_t,10)

hacspec speclib

Refinements

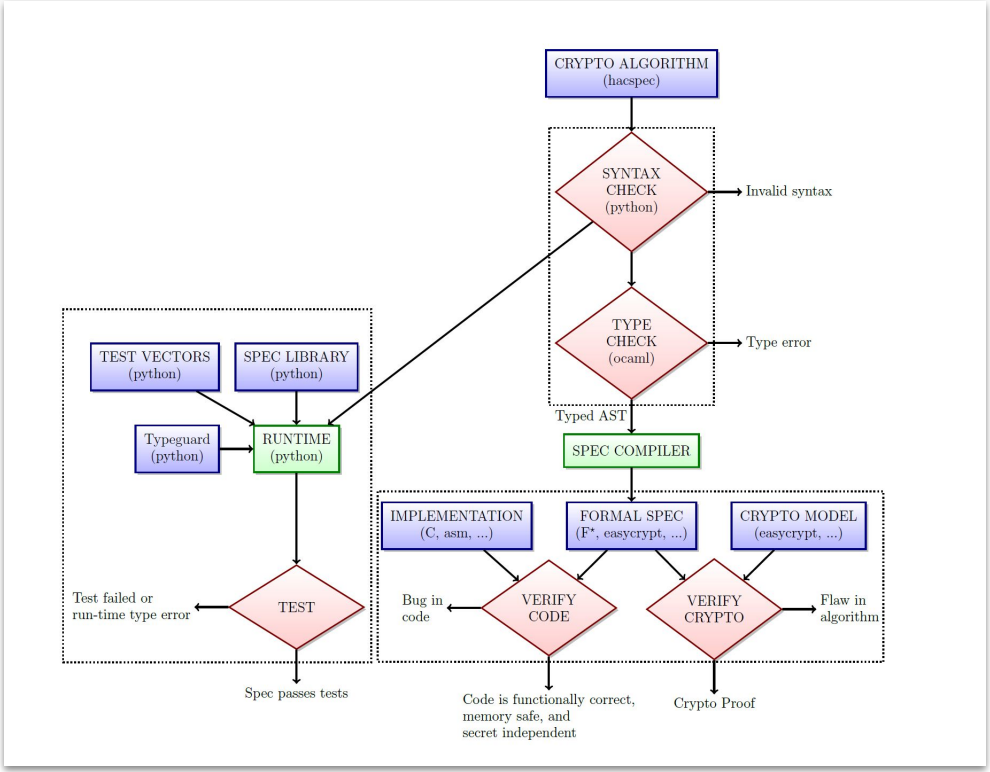
```
index_t =
  refine_t(int,
    lambda x: x < 16 and x >= 0)
```

hacspec speclib

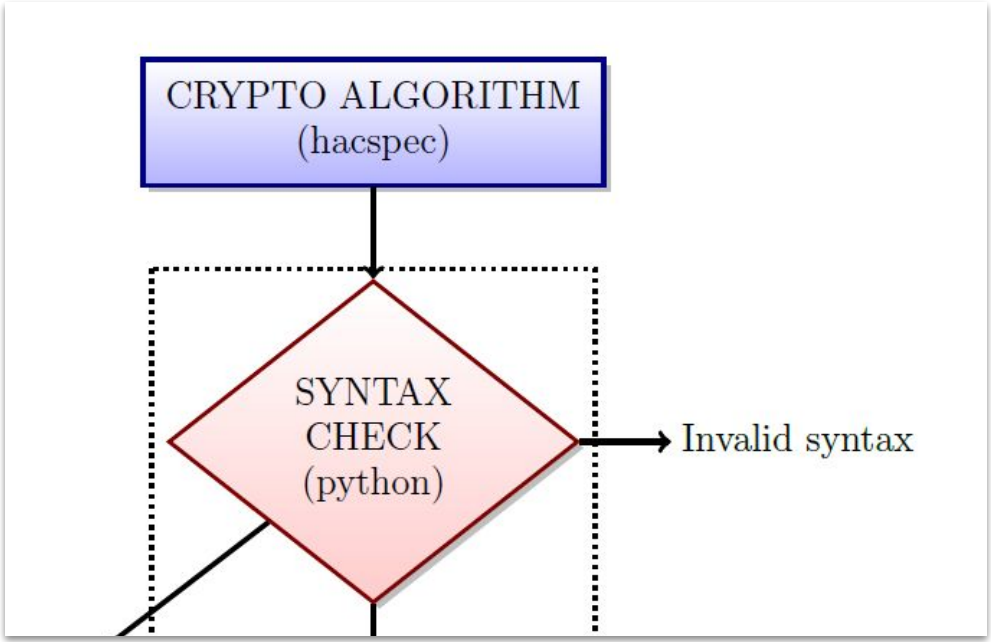
Contracts

```
@contract(
    lambda input, l: True,
    lambda input, l, res:
            array.length(res) == l
)
```

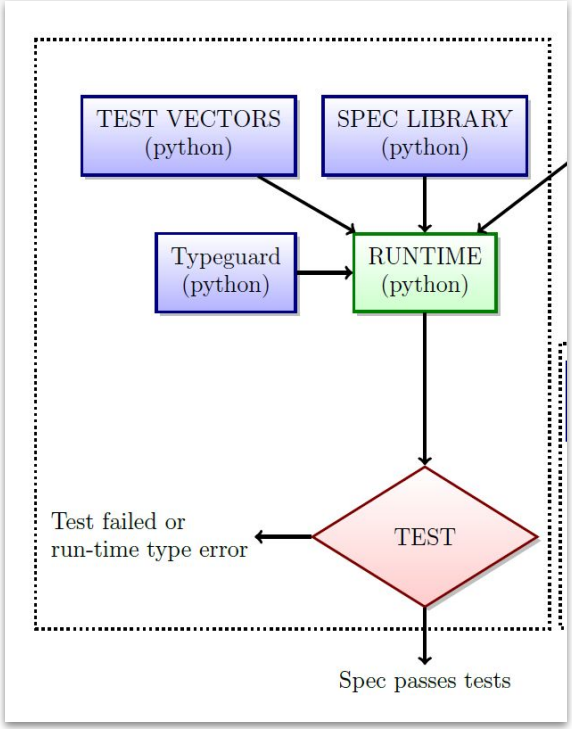
Architecture



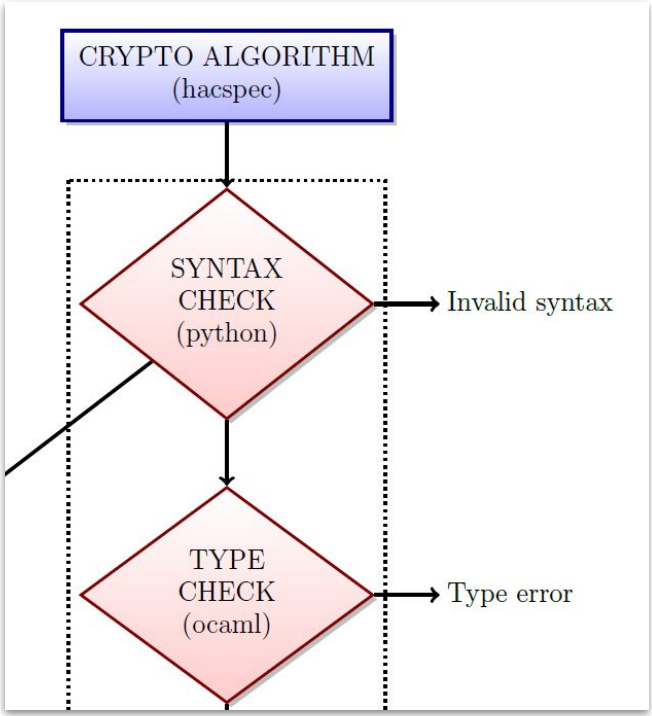
Architecture – The Spec



Architecture – Python Runtime

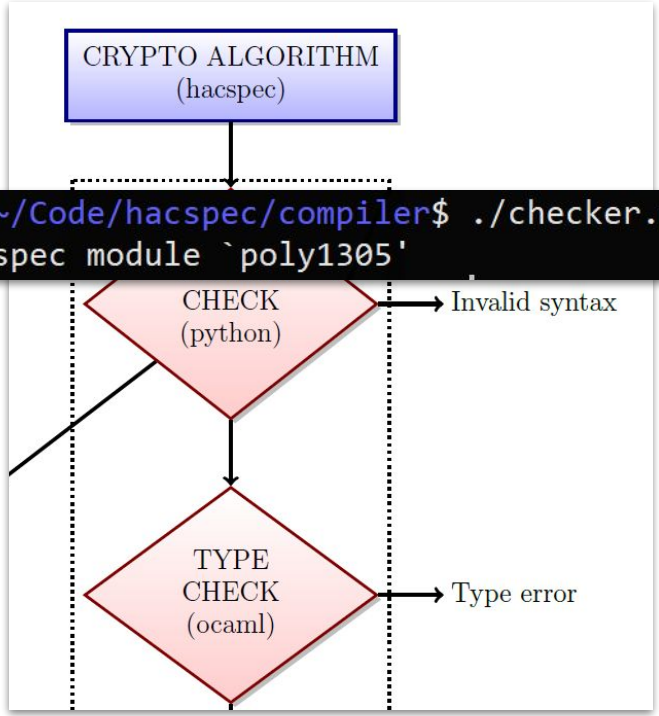


Architecture — Checker

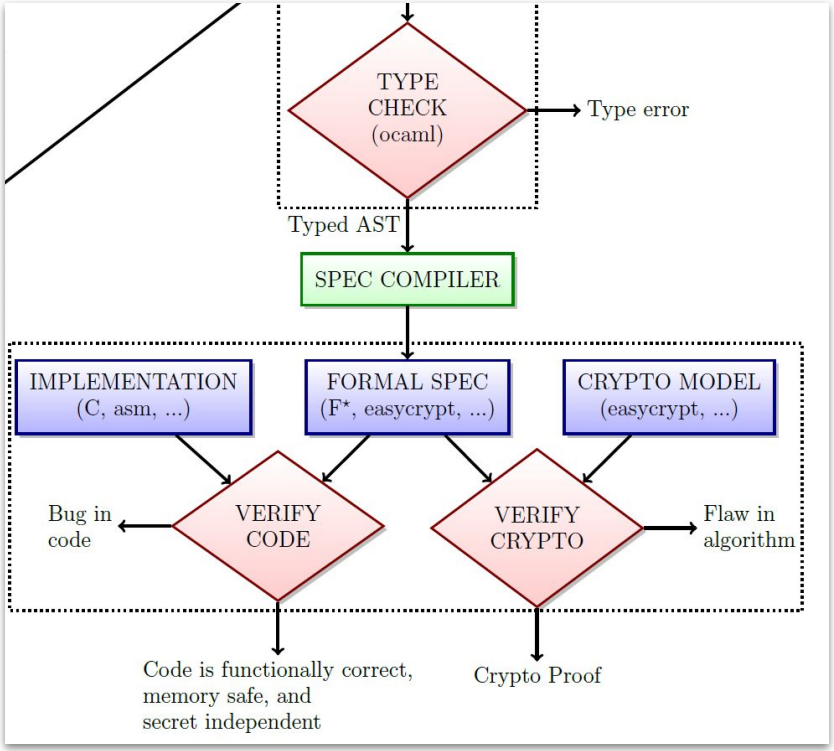


Architecture — Checker

```
franziskus@DESKTOP-FCM7CIP:~/Code/hacspec/compiler$ ./checker.native ../specs/poly1305.py
Parsed and type-checked hacspec module `poly1305'
```



Architecture — F* Compiler



F* Compiler

```
franziskus@DESKTOP-FCM7CIP:~/Code/hacspec/compiler$ HACL_HOME=~/Code/hacl-star/ FSTAR_HOME=~/Code/hacl-star
/dependencies/FStar/ make -C fstar-compiler/specs/ poly1305.fst.checked
make: Entering directory '/mnt/c/Users/Franziskus/Code/hacspec/compiler/fstar-compiler/specs'
../../to_fstar.native ../../../specs/poly1305.py > poly1305_pre.fst
/home/franziskus/Code/hacl-star/dependencies/FStar//bin/fstar.exe --include /home/franziskus/Code/hacl-star
//lib --include /home/franziskus/Code/hacl-star//lib/fst --expose_interfaces --indent poly1305_pre.fst > po
ly1305.fst
rm poly1305_pre.fst
/home/franziskus/Code/hacl-star/dependencies/FStar//bin/fstar.exe --include /home/franziskus/Code/hacl-star
//lib --include /home/franziskus/Code/hacl-star//lib/fst --expose_interfaces poly1305.fst
Verified module: Poly1305 (4871 milliseconds)
All verification conditions discharged successfully
make: Leaving directory '/mnt/c/Users/Franziskus/Code/hacspec/compiler/fstar-compiler/specs'
```

Example

Poly1305

```
p := 2^130-5
r := clamped key

for i=1 upto ceil(msg length in bytes / 16)
   n = (msg[((i-1)*16)..(i*16)] | [0x01])
   a += n
   a = (r * a) % p
   end
```

Poly1305 - hacspec

```
p = nat((2 ** 130) - 5)
felem_t = natmod_t(p)

def poly(text:vlarray_t(felem_t),
         key:felem_t) -> felem_t:
  result = natmod(0,p)
  for i in range(array.length(text)):
    result = key * (result + text[i])
  return result
```

Poly1305 - F*

```
let poly (text:vlbytes_t)(r:felem_t):felem_t =
  let acc = felem 0 in
  let acc = repeati (array_length blocks)
   (fun i acc ->
     (acc +. (encode blocks.[ i ])) *. r) acc in
  acc
```

Tested specs

Algorithm	Valid (py)	Valid	F* compilation	F* type checked
Poly1305	✓	✓	✓	✓
ChaCha20	✓	✓	✓	✓
AEAD ChaCha20Poly1305	✓	✓	✗	✗
AES	✓	✓	✓	✗
GF128	✓	✓	✓	✓
AEAD AES-GCM 128	✓	✓	✗	✗
Sha2	✓	✗	✗	✗
Sha3	✓	✗	✗	✗
Curve25519	✓	✓	✓	✗
Cuve448	✓	✓	✓	✗
ED25519	✓	✗	✗	✗
P256	✓	✗	✗	✗
RSA PSS	✓	✗	✗	✗
Blake2	✓	✗	✗	✗
WOTS	✓	✗	✗	✗
Frodo	✗	✗	✗	✗
Argon2i	✗	✗	✗	✗

Some issues

- Another compiler is needed
- Proofs might get harder
- Performance

Summary

- hacspec language
  - Testable
  - Easy to use
- Tooling available
  - Type checking
  - Compiler to formal languages

Come and help us

- **Promote** and **use** hacspec in specifications
- Give us **feedback** on the language and tooling

https://hacs-workshop.github.io/hacspec/