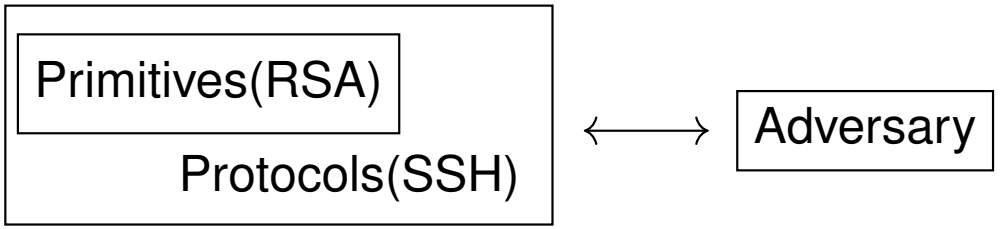
Latest improvments in the Jasmin compiler and protection against Spectre attacks

Benjamin Grégoire and Swarn Priya

Provable cryptography

Algorithms:

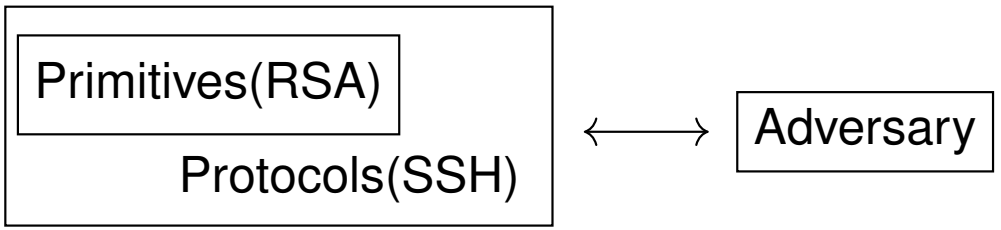
EasyCrypt



Provable security: $Pr[A \text{ breaks } P] \leq Pr[B(A) \text{ breaks assumption}] + \epsilon$

Provable cryptography

Algorithms: **EasyCrypt**



Provable security: $Pr[A \text{ breaks } P] \leq Pr[B(A) \text{ breaks assumption}] + \epsilon$

Source code: **Jasmin**

Provable security: Algorithms + Functional correctness + Safety

Provable cryptography

Algorithms: EasyCrypt

Primitives(RSA)

Protocols(SSH)

$\longleftrightarrow$ Adversary

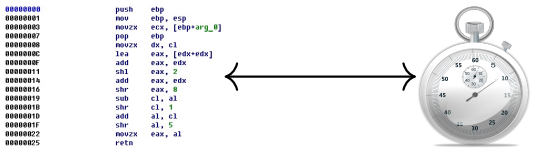
Provable security: $Pr[A \text{ breaks } P] \leq Pr[B(A) \text{ breaks assumption}] + \epsilon$

Source code: Jasmin

Code $\longleftrightarrow$ Adversary

Provable security: Algorithms + Functional correctness + Safety

Assembly: Jasmin



Provable security: Source + Compiler + CCT

Jasmin language

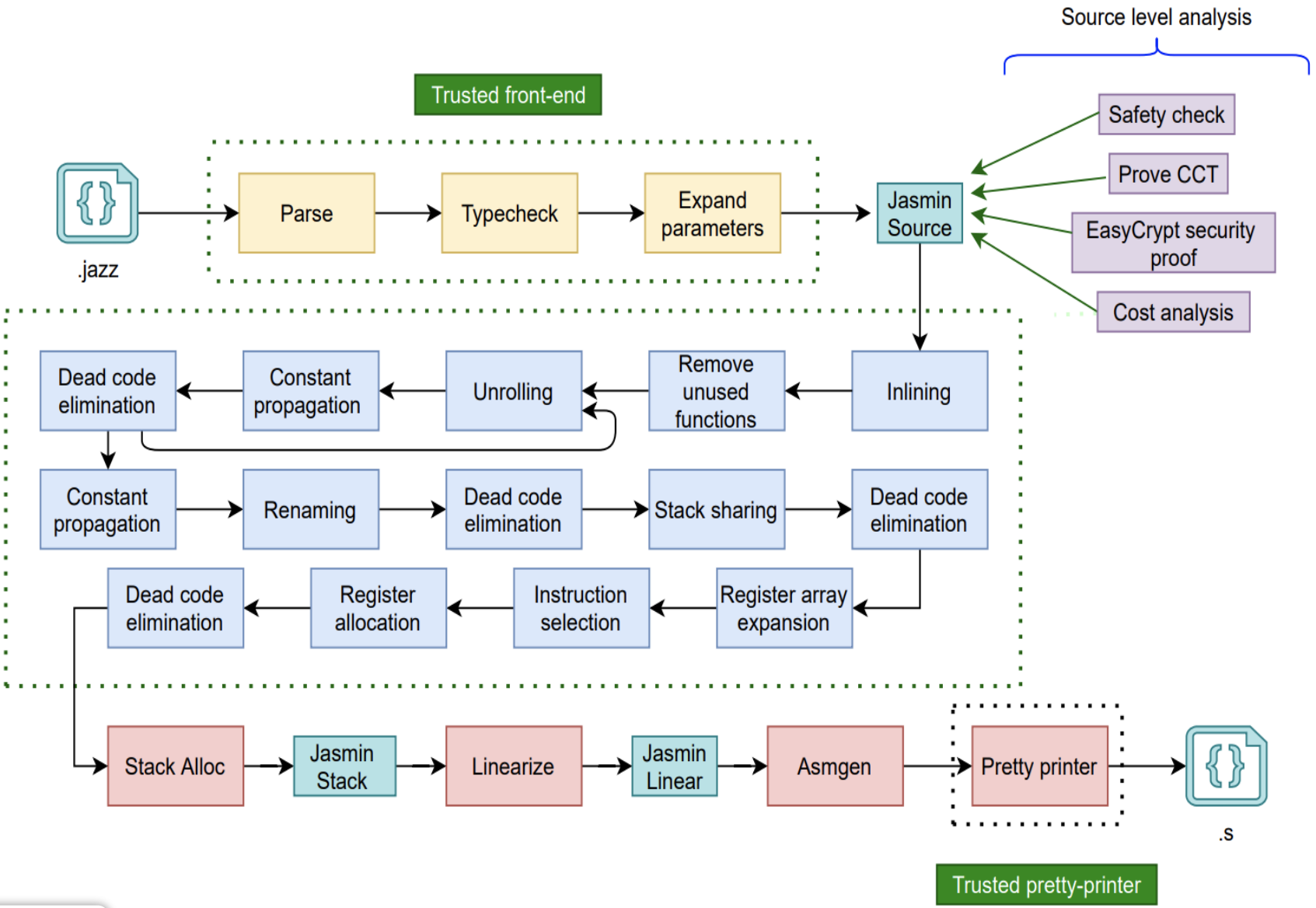
Goal: combine performance, correctness, and security of implementations

1. Combine high level abstraction (loop, array, function, . . .) and low-level (assembly instruction, register, stack)
2. A predictable compiler, formally proved in the Coq proof assistant
3. A simple and clear semantic $\Rightarrow$ automatic and interactif verification tools

A Jasmin program

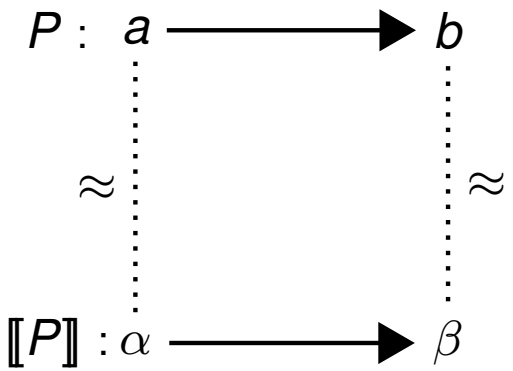
```
inline fn mix2accs(stack u64[8] acc, inline int off, reg u64 p) −→ reg u64 {
    reg u64[2] data;
    reg u64 m;
    inline int i;
    for i = 0 to 2 {
        data[i] = acc[i + off];
        data[i] ^= [p + 8 * i];
    }
    m = mul128fold64(data);
    return m;
}
```

The Jasmin compiler



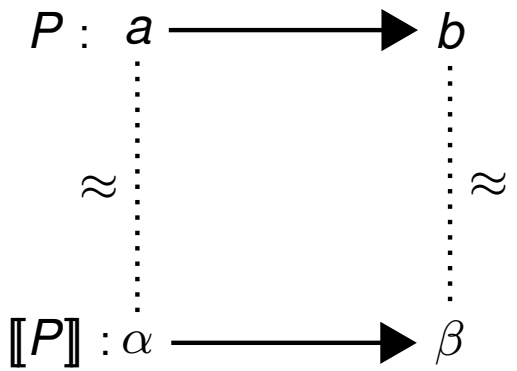
Correctness of the compiler

- *a* and *b* are source states (Jasmin level)
- α and β are target states (x86 level)



Correctness of the compiler

- a and b are source states (Jasmin level)
- α and β are target states (x86 level)



Correctness of the compiler allows to map:

- functional correctness from source to assembly
- exact security from source to assembly

Does not help for CCT

Cryptographic Constant Time (CCT)

Crypto implementation need to be protected against cache attacks

```
if (e) {
  x = p[i];
}
```

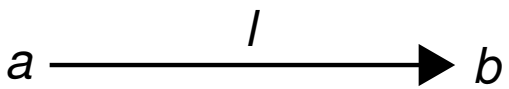
e leaks, so needs to be public

the address of p[i] leaks, so · · ·

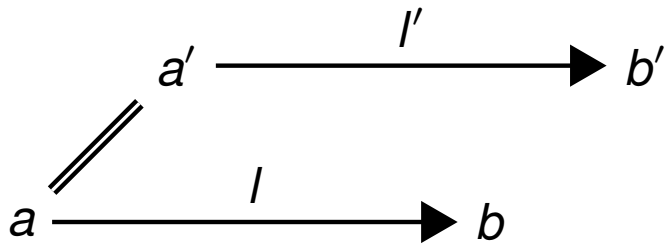
Cryptographic constant time:

- No branching on secret dependent condition
- No load/store on secret dependent address

Cryptographic Constant Time (CCT)

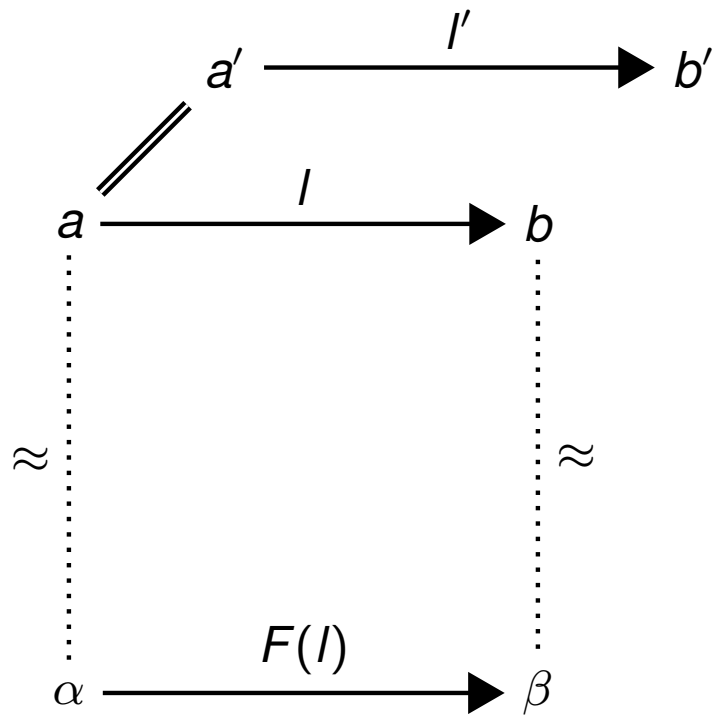


Cryptographic Constant Time (CCT)



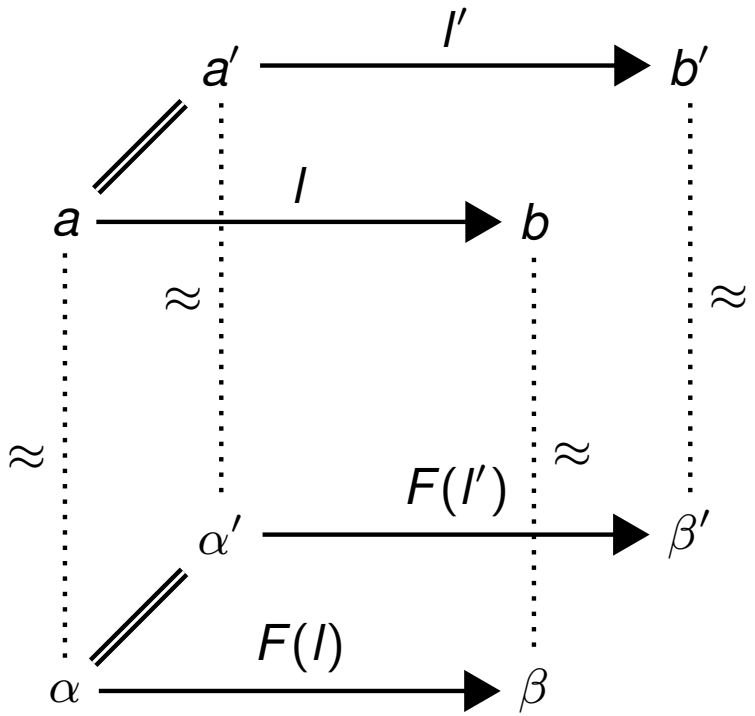
$$l = l'$$

Cryptographic Constant Time (CCT)



$$l = l'$$

Cryptographic Constant Time (CCT)



$$l = l'$$

$$F(l) = F(l')$$

Preservation of Constant Time

Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Swarn Priya: Structured Leakage and Applications to Cryptographic Constant-Time and Cost. CCS 2021:

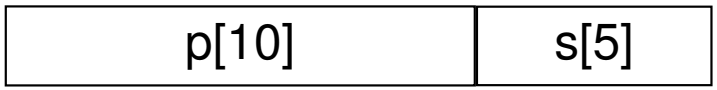
- Preservation of constant time
- Cost analysis at assembly level from cost analysis at source level

Under submission:

- Strengthening the model (div, mod)
- Weakening the model (cache line)
- theoretical attack found in openSSL (MEE-CBC) + patch provided

Problem : CCT does not protect against Specter attacks.

The problem with speculation (Specter V1)




```
// i = 11
if (i < 10) {
  x = p[i];
  if (x = 0) { ··· }
  [x] = 0;
}
```

leaks x = 0 (so (s[0] = 0)

leaks x (so s[0])

Counter measures

- use LFENCE:
  Gilles Barthe, Sunjay Cauligi, Benjamin Grégoire, Adrien Koutsos, Kevin Liao, Tiago Oliveira, Swarn Priya, Tamara Rezk, Peter Schwabe: High-Assurance Cryptography in the Spectre Era. IEEE Symposium on Security and Privacy 2021: 1884-1901
- use Speculative Load Hardening (llvm)
- use Selective Speculative Load Hardening (Jasmin, work in progress)

Where we are ?

- A type system for SSLH at source level
- Various implementations of crypto and post-quantum crypto are already protected (scalar and avx2 implementations):
  - hash algorithms: sha256, sha3-224/256/384/512, shake128/256
  - one time auth: poly1305
  - stream: chacha, salsa20, xsalsa20
  - KEM : Kyber (on going work)
- A proof of the type system for a toy language
- But we do not understand how to prove preservation of SCT . . .
- On going work:
  - A type system for SSLH at assembly level
  - The type system relies on a points-to analysis (provided by the compiler)
  - Proof in progress for Jasmin