EasyCrypt and Jasmin Tutorial

Cyber in Saclay school

Manuel Barbosa (mbb@fc.up.pt)
Benjamin Grégoire (benjamin.gregoire@inria.fr)
François Dupressoir (f.dupressoir@bristol.ac.uk)
Vincent Laporte (vincent.laporte@inria.fr)
Pierre-Yves Strub (pierre-yves@strub.nu)

February 11th 2021

What to expect

What we will cover

How to have a certified assembly implementation with the associate security proofs:

- cryptographic correctness
- cryptographic security
- cryptographic constant-time

We will use a very simple example that will allow to cover many aspects of this.

What we will use

The EasyCrypt proof assistant:

- Specify syntax and security models for crypto algorithms
- Specify crypto assumptions and concrete crypto algorithms
- Prove crypto algorithms correct and secure

What we will use

The EasyCrypt proof assistant:

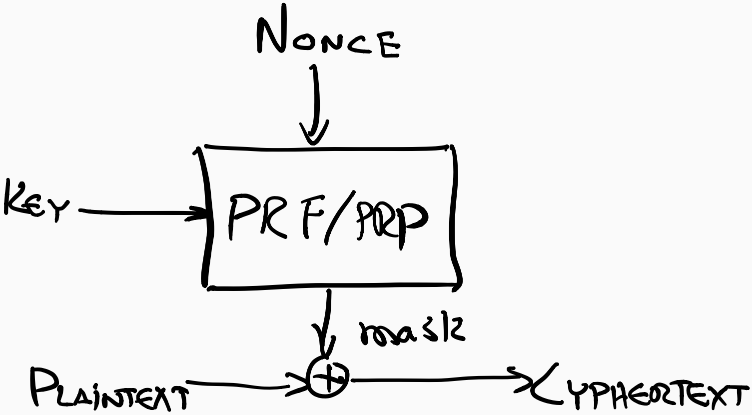
- Specify syntax and security models for crypto algorithms
- Specify crypto assumptions and concrete crypto algorithms
- Prove crypto algorithms correct and secure

The Jasmin language and compiler:

- Write high-speed crypto code and compile it to assembly
- Automatically safety-check Jasmin programs
- Extract Jasmin to EasyCrypt for correctness/security proofs
- Extract Jasmin to EasyCrypt for constant-time verification

The example

The example: textbook symmetric encryption from PRF/PRP



The example: provable security view

The construction in crypto terms

Fix:

- the key space K
- the nonce space N
- the message space M
- the ciphertext space $C := M$

Let f be a function of type $\mathsf{f} : K \times N \to M$.

Key generation: sampling uniformly at random from K

Encryption: $\mathsf{Enc}(k, n, m) := m \oplus \mathsf{f}(k, n)$

Decryption: $\mathsf{Dec}(k, n, c) := c \oplus \mathsf{f}(k, n)$

(Nonce-based) IND\$-CPA security

Game $\text{IND\$-CPA-Real}_{\mathcal{A}}(\,)$

$k \twoheadleftarrow K$

$b \twoheadleftarrow \mathcal{A}^{\text{RealEnc}(\cdot,\cdot)}(\,)$

Return b

proc $\text{RealEnc}(n, m)$

Return $\text{Enc}(k, n, m)$

Game $\text{IND\$-CPA-Ideal}_{\mathcal{A}}(\,)$

$b \twoheadleftarrow \mathcal{A}^{\text{IdealEnc}(\cdot,\cdot)}(\,)$

Return b

proc $\text{IdealEnc}(n, m)$

$c \twoheadleftarrow C$

Return c

Security requires the following advantage measure to be small

$$\text{Adv}_{\text{CPA}}(\mathcal{A}) = \\ \left|\, \Pr\left[\, \text{IND\$-CPA-Real}_{\mathcal{A}}(\,) \Rightarrow \text{true} \,\right] - \Pr\left[\, \text{IND\$-CPA-Ideal}_{\mathcal{A}}(\,) \Rightarrow \text{true} \,\right] \right|$$

Pseudorandom Functions

Let f be a function of type $f : K \times N \to M$.

Game PRF-Real$_{\mathcal{A}}$( )

$k \twoheadleftarrow K$

$b \twoheadleftarrow \mathcal{A}^{f(k,\cdot)}()$

Return b

Game PRF-Ideal$_{\mathcal{A}}$( )

$T \leftarrow \{\,\}$

$b \twoheadleftarrow \mathcal{A}^{F(\cdot)}()$

Return b

proc F(x):

If $x \notin T$: $T[x] \twoheadleftarrow M$

Return $T[x]$

F is a truly random function (lazily sampled).

f is pseudorandom if the following advantage measure is small

$$\mathrm{Adv}_{\mathrm{PRF}}(\mathcal{A}) = |\Pr[\text{PRF-Real}_{\mathcal{A}}() \Rightarrow \text{true}] - \Pr[\text{PRF-Ideal}_{\mathcal{A}}() \Rightarrow \text{true}]|$$

Restrictions on attacker power

We will prove that forall $\mathcal{A}$: $\mathsf{Adv}_{\mathsf{CPA}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}(\mathcal{A}))$

Restrictions on attacker power

We will prove that forall $\mathcal{A}$: $\mathsf{Adv}_{\mathsf{CPA}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}(\mathcal{A}))$

Restrictions that come up explicitly in EasyCrypt:

- Do *not* place two queries with the same nonce n
- Place at most q oracle queries

Restrictions on attacker power

We will prove that forall $\mathcal{A}$: $\mathsf{Adv}_{\mathsf{CPA}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{B}(\mathcal{A}))$

Restrictions that come up explicitly in EasyCrypt:

- Do *not* place two queries with the same nonce n
- Place at most q oracle queries

Restrictions on attacker power that will be implicit:

- IND$-CPA attacker executes in at most t steps
- we assume that PRF/PRP cannot be broken in $\sim t$ steps

Those restrictions are not needed to prove the exact security bound, they are only necessary to prove that the advantage is negligible

Easycrypt formalization

The example: implementation view

Jasmin: last mile of high assurance cryptography

We want **fast** and **formally verified** assembly code:

- Source language: control on the generated assembly + formal semantics

  $\Longrightarrow$ programmer & verification friendly

- Compiler: predictable & formally verified (in Coq)

  $\Longrightarrow$ programmer has control and no compiler security bug

- Verification toolchain (based Easycrypt):
  - safety
  - functional correctness
  - security
  - constant-time

Jasmin language / Assembly:

Jasmin is a relatively high level language:

- Variable, array, loop, function call,
- Simple semantic (no alias)

Jasmin provides a strong control on the generated assembly:

- Variable can be tagged as reg/stack
- Access to low level assembly instructions and flags
- Control on loop unrolling: for vs while
- Inlining introduces no extra instruction
- Control over constant propagation/partial evaluation:

  inline variables

Jasmin-code and link with Easycrypt

Take-aways

Main take-aways on Jasmin

Using Jasmin for writing high-speed code:

+ It is a new language for optimized low-level code

+ Programming in Jasmin requires no knowledge of verification

+ Safety of Jasmin programs checked automatically

– Currently we only support x86-64 platforms

Main take-aways on Jasmin compiler

+ The Jasmin compiler is proved in Coq:

⇒ Functional correctness can be propagate to assembly

+ On going work for preservation of constant time:

⇒ This is not ensured by correctness of the compiler

Main take-aways on Jasmin compiler

$+$ The Jasmin compiler is proved in Coq:

$\Rightarrow$ Functional correctness can be propagate to assembly

$+$ On going work for preservation of constant time:

$\Rightarrow$ This is not ensured by correctness of the compiler

Jasmin correctness and constant-time:

$+$ Jasmin correctness in EasyCrypt $=$ standard Hoare logic

$+$ Jasmin CT in EasyCrypt $=$ mostly automatic

Main take-aways on EasyCrypt

+ Specifying crypto in EC requires no knowledge of verification

+ Specifying game-hops in EC requires no knowledge of verification

– Proofs are not automatic, although some automation exists

– Multidisciplinary team required for getting end-to-end results

Thank you for attending!