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

```
NONCE

PRF/PRP

Key

Plaintext

\oplus

\text{mask}

\oplus

Ciphertext
```
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 $f : K \times N \rightarrow M$.

Key generation: sampling uniformly at random from $K$

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

Decryption: $\text{Dec}(k, n, c) := c \oplus f(k, n)$
(Nonce-based) IND$-CPA security

\[
\begin{align*}
\text{Game } \text{IND$-$CPA-Real}_A(\cdot) & \quad \text{Game } \text{IND$-$CPA-Ideal}_A(\cdot) \\
\begin{align*}
& k \leftarrow K \\
& b \leftarrow A^{\text{RealEnc}(\cdot, \cdot)}(\cdot) \\
\end{align*} \\
& \text{Return } b \\
& \text{proc } \text{RealEnc}(n, m) \\
& \quad \text{Return } \text{Enc}(k, n, m) \\
& b \leftarrow A^{\text{IdealEnc}(\cdot, \cdot)}(\cdot) \\
& \text{Return } b \\
& \text{proc } \text{IdealEnc}(n, m) \\
& \quad c \leftarrow C \\
& \quad \text{Return } c
\end{align*}
\]

Security requires the following advantage measure to be small

\[
\text{Adv}_{\text{CPA}}(A) = \left| \Pr[\text{IND$-$CPA-Real}_A(\cdot) \Rightarrow \text{true}] - \Pr[\text{IND$-$CPA-Ideal}_A(\cdot) \Rightarrow \text{true}] \right|
\]
Let $f$ be a function of type $f : K \times N \rightarrow M$.

**Game PRF-Real$_A$():**
- $k \leftarrow K$
- $b \leftarrow A^{f(k,\cdot)}()$
- Return $b$

**Game PRF-Ideal$_A$():**
- $T \leftarrow \{\}$
- $b \leftarrow A^{F(\cdot)}()$
- Return $b$

**proc F(x):**
- If $x \notin T$: $T[x] \leftarrow M$
- Return $T[x]$

$F$ is a truly random function (lazily sampled).

$f$ is pseudorandom if the following advantage measure is small

$$\text{Adv}_{\text{PRF}}(A) = | \Pr[\text{PRF-Real}_A() \Rightarrow \text{true}] - \Pr[\text{PRF-Ideal}_A() \Rightarrow \text{true}] |$$
Restrictions on attacker power

We will prove that for all \( \mathcal{A} \): \( \text{Adv}_{\text{CPA}}(\mathcal{A}) = \text{Adv}_{\text{PRF}}(\mathcal{B}(\mathcal{A})) \)
We will prove that for all $\mathcal{A}$: $\text{Adv}_{\text{CPA}}(\mathcal{A}) = \text{Adv}_{\text{PRF}}(\mathcal{B}(\mathcal{A}))$

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.
We will prove that for all $A$: $\text{Adv}_{\text{CPA}}(A) = \text{Adv}_{\text{PRF}}(B(A))$

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

- **Source language**: control on the generated assembly + formal semantics
  
  $$\Rightarrow$$ programmer & verification friendly

- **Compiler**: predictable & formally verified (in Coq)
  
  $$\Rightarrow$$ programmer has control and no compiler security bug

- **Verification toolchain (based Easycrypt)**:
  
  - safety
  - functional correctness
  - security
  - constant-time
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:
  ⇒ Functional correctness can be propagate to assembly
+ On going work for preservation of constant time:
  ⇒ 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!