Interactive theorem proving for protocol verification

Horațiu Cheval [1,2]

[1]University of Bucharest

[2]Institute for Logic and Data Science

Dresden, March 28, 2024

Interactive theorem provers

- Software that allows one to state, in principle, any mathematical statement and to provide a proof thereof, which will be automatically checked for correctness.
- Agda, Coq, Isabelle, Lean etc.
- In general, a small trusted code base (the kernel)

Lean

- An ITP based on dependent type theory
- A fully-fledged functional programming language
- Rich metaprogramming capabilities
- A large corpus of formalized mathematics in its `mathlib` library
- So far, more mathematical applications than CS-related (not to say that they do not exist)

Semantic security

Some formalizations include:

- Nowak's framework [1], Foundational Cryptography Framework [2], EasyCrypt [3], SSProve [4] (Coq)
- crypto-agda (Agda) [5]
- CryptHOL (Isabelle/HOL) [6]
- cryptolib (Lean 3) [7]

The examples we present in the following are from cryptolib and are part of a WIP translation of cryptolib to Lean 4.

Semantic security

- A challenger C generates a public key pk
- The attacker A produces two messages m_1, m_2
- C chooses a message m_i, encrypts it with pk and sends it to A
- A's task is to determine which of the two messages was encrypted

Then, A wins the game if it determines the correct m_i, and the semantic security properties states that the probability of this happing is negligibly close to $\frac{1}{2}$.

Semantic security

```
variable {PKey SKey Message Cypher S : Type}
  (keygen : PMF (PKey × SKey))
  (encrypt : PKey → Message → PMF Cypher)
  (decrypt : SKey → Cypher → Message)
  (attacker : PKey → PMF (Message × Message × S))
  (attacker' : Cypher → S → PMF ℤ₂)

def semanticSecurityGame : PMF ℤ₂ := do
  let (pk, sk) ← keygen
  let (m₁, m₂, s) ← attacker pk
  let b ← PMF.uniformOfFintype ℤ₂
  let cypher ← encrypt pk
    (if b = 0 then m₁ else m₂)
  let b' ← attacker' cypher s
    return (if b = b' then 1 else 0)
```

Semantic security

The two main properties of a protocol are formalized as:

```
def SemanticSecurity (ε : NNReal) : Prop :=
  let p := SSG keygen encrypt attacker attacker’ 1
  abs (p.toReal - 1/2) ≤ ε
```

```
def Corectness : Prop := ∀ m,
  encryptDecrypt keygen encrypt decrypt m = pure 1
```

ElGamal

A public key encryption algorithm (in particular ElGamal) is thus specified by providing concrete definitions for the `keygen`, `encrypt` and `decrypt` functions. For ElGamal, we need to assume a finite group G and a generator g of G.

```
variable
  (G : Type) [Fintype G] [CommGroup G]
  (g : G) (hg : ∀ x : G, x ∈ Subgroup.zpowers g)
```

The correctness is proved using arithmetic in finite groups, and reasoning about equality on pmf's, relying heavily on mathlib.

```lean
lemma decrypt_eq_m (m : G) (x y: ZMod q) : decrypt x ((g^y.val), ((g^x.val)^y.val * m)) = m := by
  simp [decrypt]
  rw [← pow_mul g x.val y.val]
  rw [← pow_mul g y.val x.val]
  rw [mul_comm y.val x.val]
  aesop?
```

▼ LeanProject.lean:88:4

▼ Tactic state

1 goal

```
G : Type
inst✝² : Fintype G
inst✝¹ : CommGroup G
inst✝ : DecidableEq G
g : G
hg : ∀ (x : G), x ∈ Subgroup.zpowers g
S : Type
attacker : G → PMF (G × G × S)
attacker' : G → G → S → PMF ℤ₂
m : G
x y : ZMod q
⊢ decrypt x (g ^ ZMod.val y, (g ^ ZMod.val x) ^ ZMod.val y * m) = m
```

```
lemma decrypt_eq_m (m : G) (x y: ZMod q) : decrypt x ((g^y.val), ((g^x.val)^y.val * m)) = m := by
  simp [decrypt]
  rw [← pow_mul g x.val y.val]
  rw [← pow_mul g y.val x.val]
  rw [mul_comm y.val x.val]
  aesop?
```

▼ LeanProject.lean:88:18

▼ Tactic state

1 goal

```
G : Type
inst✝² : Fintype G
inst✝¹ : CommGroup G
inst✝ : DecidableEq G
g : G
hg : ∀ (x : G), x ∈ Subgroup.zpowers g
S : Type
attacker : G → PMF (G × G × S)
attacker' : G → G → S → PMF ℤ₂
m : G
x y : ZMod q
⊢ (g ^ ZMod.val x) ^ ZMod.val y * m / (g ^ ZMod.val y) ^ ZMod.val x = m
```

```lean
lemma decrypt_eq_m (m : G) (x y: ZMod q) : decrypt x ((g^y.val), ((g^x.val)^y.val * m)) = m := by
  simp [decrypt]
  rw [← pow_mul g x.val y.val]
  rw [← pow_mul g y.val x.val]
  rw [mul_comm y.val x.val]
  aesop?
```

▼ LeanProject.lean:89:32

▼ Tactic state

1 goal

```
G : Type
inst✝² : Fintype G
inst✝¹ : CommGroup G
inst✝ : DecidableEq G
g : G
hg : ∀ (x : G), x ∈ Subgroup.zpowers g
S : Type
attacker : G → PMF (G × G × S)
attacker' : G → G → S → PMF ℤ₂
m : G
x y : ZMod q
⊢ g ^ (ZMod.val x * ZMod.val y) * m / (g ^ ZMod.val y) ^ ZMod.val x = m
```

```
87 lemma decrypt_eq_m (m : G) (x y: ZMod q) : decrypt x ((g^y.val), ((g^x.val)^y.val * m)) = m := by
88   simp [decrypt]
89   rw [← pow_mul g x.val y.val]
90   rw [← pow_mul g y.val x.val]
91   rw [mul_comm y.val x.val]
92   aesop?
93
94
```

▼ LeanProject.lean:92:10

▼ Tactic state

No goals

▼

▼ Suggestions

```
Try this:
      rename_i
      inst
      inst_1
      inst_2
    simp_all only [mul_div_cancel_left]
```

ElGamal

Finally, the main result uses game hopping to show that that the *decisional Diffie-Hellman assumption*, stating (g^x, g^y, g^z) and (g^x, g^y, g^{xy}), where $x, y, z \in \mathbb{Z}_q$ are picked uniformly, cannot be distinguished, implies semantic security.

The DDH assumption can be readily expressed in Lean, similarly as before, using

```
def DDH₀ : PMF ℤ₂ := do
    let x ← PMF.uniformOfFintype (ZMod q)
    let y ← PMF.uniformOfFintype (ZMod q)
    D (g^x.val) (g^y.val) (g^(x.val * y.val))
def DDH₁ : PMF ℤ₂ := do
    let x ← PMF.uniformOfFintype (ZMod q)
    let y ← PMF.uniformOfFintype (ZMod q)
    let z ← PMF.uniformOfFintype (ZMod q)
    D (g^x.val) (g^y.val) (g^z.val)
```

ElGamal

The main theorem (we won't go into the proof):

```
theorem ElGamal.SemanticSecurity :
  DDH g hg →
  SemanticSecurity
    ElGamal.keygen
    ElGamal.encrypt
    attacker attacker' ε := ...
```

Thank you

References

[1] D. Nowak (2007). A framework for game-based security proofs. In Information and Communications Security: 9th International Conference, ICICS 2007, Zhengzhou, China, December 12-15, 2007. Proceedings 9 (pp. 319-333). Springer Berlin Heidelberg.

[2] A. Petcher (2015). A Foundational Proof Framework for Cryptography. Harvard University.

[3] G. Barthe, B. Grégoire, S. Heraud, S.Z. Béguelin (2011). Computer-aided security proofs for the working cryptographer. In Annual Cryptology Conference (pp. 71-90). Berlin, Heidelberg: Springer Berlin Heidelberg.

[4] C. Abate, P. Haselwarter, E. Rivas, A. Van Muylder, T. Winterhalter, C. Hrițcu, K. Maillard, B.Spitters (2021). SSprove: A foundational framework for modular cryptographic proofs in Coq. In 2021 IEEE 34th Computer Security Foundations Symposium (CSF) (pp. 1-15). IEEE.

References

[5] D. Gustafsson, N. Pouillard (2011). Dependent protocols for communication.
[6] D. A. Basin, A. Lochbihler, S.R. Sefidgar (2020). CryptHOL: Game-based proofs in higher-order logic. Journal of Cryptology, 33, 494-566.
[7] J. Lupo (2021). cryptolib: Security Proofs in the Lean Theorem Prover. University of Edinburgh.