# Joclean

Final project for PROOF101 course

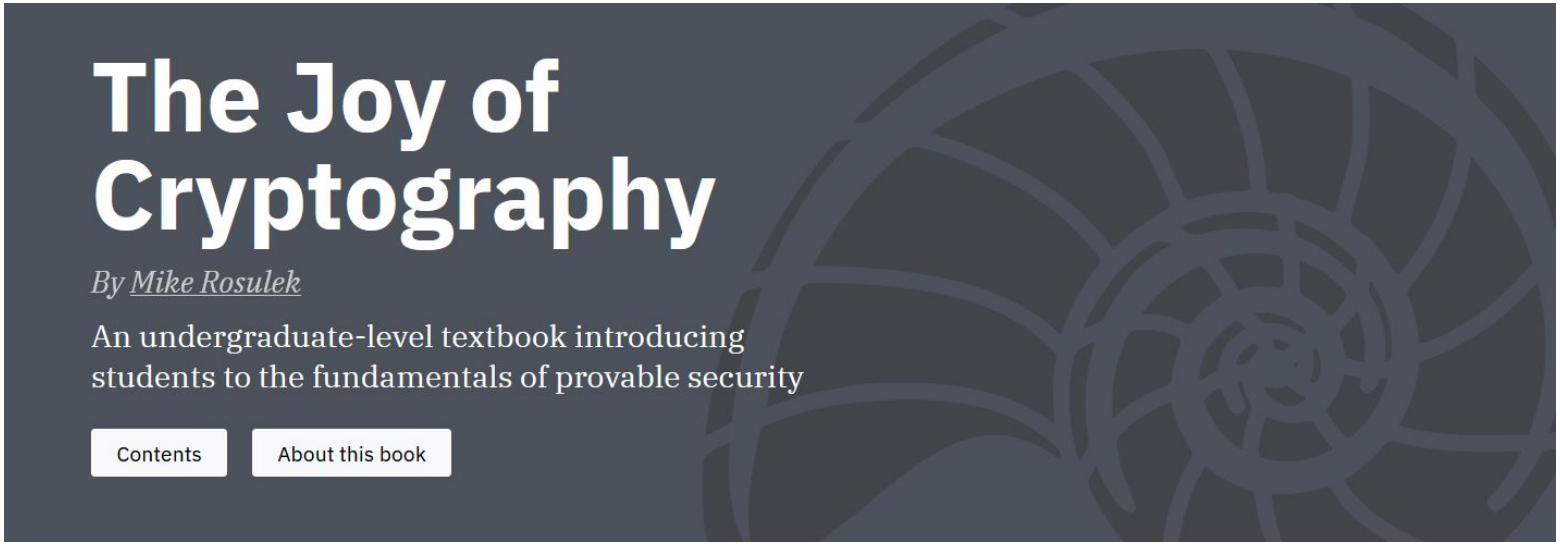
**Paweł Woźniak**

# Who Am I

- Poznań, Poland (Central Europe)
- Recently defended master’s thesis on isogeny-based cryptography
- Computer Science
- Interested in cryptography
- Found out about the course from social media

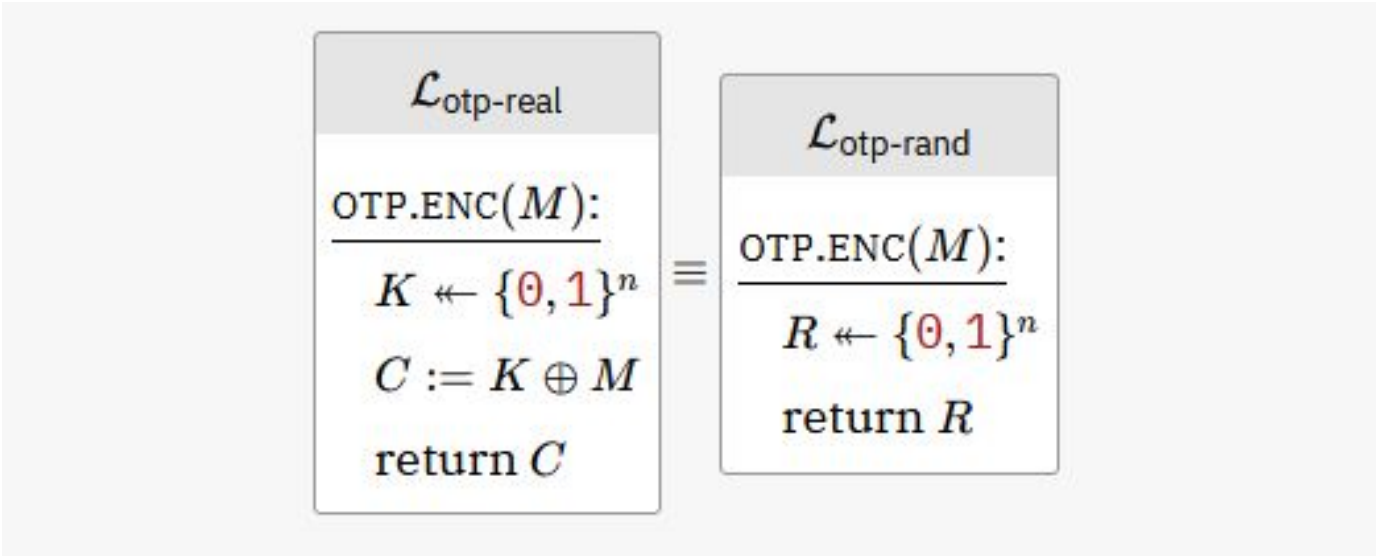
# The Joy of Cryptography

- Book about provable cryptography
- Library-based approach for conducting proofs
- 2nd edition released this year, 3 first chapters available online



# The Joy of Cryptography

- Claim 1.4.1 - OTP encryption is uniformly distributed
- Notion of "**indistinguishability**" between libraries (Claim 2.2.2)



# Joclean - (very early) Formalization of JoC

- **Proposal**:
  Design *Interface*, *Library* and *Adversary* with tactics

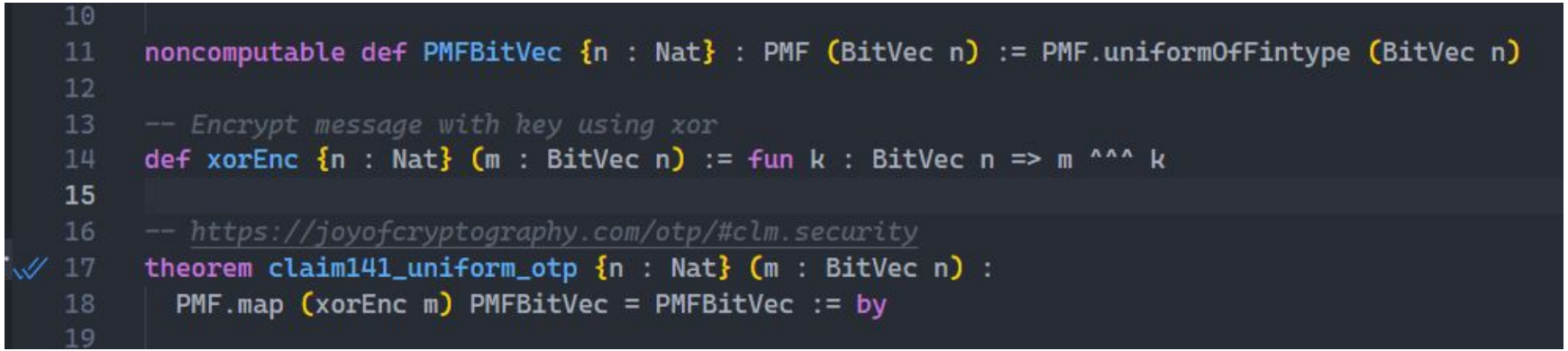
- **What I did**:
  Formalize the OTP 1.4.1 uniformity claim **without** introducing the "Library" and "indistinguishability" relation (**yet**)

- **Future Work:**
  Finish the idea from proposal - wrap the OTP uniformity claim into the Library framework

Link: https://github.com/desecnd/proof101-project

# Joclean - OTP Claim 1.4.1

- BitVec - Lean4 builtin for n-bit string operations
- PMF - Probability Mass Function (uniform distribution for fin types)
- *Encrypting the message M with uniform key, gives a uniform BitVec*



# Joclean - OTP Claim 1.4.1

- Convert into a `tsum` object

```
19
20    -- rw? => change map to the "for all" qualitifier
21    rw [PMF.ext_iff]
22
23    -- introduce a new ciphertext bitvec
24    intro c
25
26    -- rw? => change map into tsum object over probabilities
27    rw [PMF.map_apply]
```

```
1 goal
n : ℕ
m c : BitVec n
⊢ (Σ' (a : BitVec n), if c = xorEnc m a then PMFBitVec a else 0) = PMFBitVec c
```

# Joclean - OTP Claim 1.4.1

- Split into 2 cases by the `if-then-else`

```
40  let k' := c ^^^ m
41
42  -- Useful to get the the c = m ^^^ k' based on the xorEnc
43  have h_k'_eq_m_xor_c : c = xorEnc k' m := by
44    unfold xorEnc
45    simp only [k']
46    rw [BitVec.xor_assoc, BitVec.xor_self, BitVec.xor_zero]
47
48  rw [tsum_eq_single k']
```

```
h_k'_eq_m_xor_c : c = xorEnc k' m
⊢ (if c = xorEnc m k' then PMFBitVec k' else 0) = PMFBitVec c
```

```
h_k'_eq_m_xor_c : c = xorEnc k' m
⊢ ∀ (b' : BitVec n), b' ≠ k' → (if c = xorEnc m b' then PMFBitVec b' else 0) = 0
```

# Joclean - OTP Claim 1.4.1

- Solve goal 1

```
h_k'_eq_m_xor_c : c = xorEnc k' m
⊢ (if c = xorEnc m k' then PMFBitVec k' else 0) = PMFBitVec c
```

```
50    -- First Goal: we have k = k'
51    · simp only [h_k'_eq_m_xor_c]
52      unfold xorEnc
53      rw [BitVec.xor_comm]
54
55      -- PMFBitVec k_valid = PMFBitVec (m ^^^ k_valid)
56      -- Probabilities of single vectors are the same, simp works here
57      simp [PMFBitVec]
```

# Joclean - OTP Claim 1.4.1

- Solve goal 2 - by contradiction

```
h_k'_eq_m_xor_c : c = xorEnc k' m
⊢ ∀ (b' : BitVec n), b' ≠ k' → (if c = xorEnc m b' then PMFBitVec b' else 0) = 0
```

```
59  -- Second Goal: we have k != k'
60  · intro k h_neq
61    simp only [ite_eq_right_iff, h_k'_eq_m_xor_c]
62
63    -- We want to get from: xorEnc k' m = xorEnc m k
64    -- To: k = k' to prove by exfalso
65
66    -- a) We unfold xorEnc to ^^^ on BitVec:  k' ^^^ m = m ^^^ k
67    -- b) We reorder the arguments:           m ^^^ k' = m ^^^ k
68    -- c) We eliminate the xor by m from both sides, using `xor_right_inj` like in NNG4
69    unfold xorEnc
70    rw [BitVec.xor_comm, BitVec.xor_right_inj m]
71
72    -- We introduce k' = k'
73    intro h_keys_equal
74
75    -- We have both: k = k' and k ≠ k', we can prove by exfalso
76    exfalso
77
78    -- We can finish with P and ¬P, we use .symm to fit
79    exact h_neq h_keys_equal.symm
```

# Joclean - Future Work

- Understand better how `OracleComp` works in `VCVio` (Game-based proofs used for cryptographic protocols)
- Think how to adjust this type (if possible) or create something similar for the library-based proofs

```
16    /-- `OracleComp spec α` represents computations with oracle access to oracles in `spec`,
17    where the final return value has type `α`, represented as a free monad over the `PFunctor`
18    corresponding to `spec.` -/
19    def OracleComp {ι : Type u} (spec : OracleSpec.{u, v} ι) :
20        Type w → Type (max u v w) :=
21      PFunctor.FreeM spec.toPFunctor
```

# Thanks!

Thanks to organizers of PROOF101 and especially to **Daniel Dia** for creating this amazing course!