1) Primitives: KEM-DEM Security Pen & Paper Proof
2) KEM-DEM & more in ProofFrog

CAPS 2025
https://prooffrog.github.io/
https://eprint.iacr.org/2025/418

Douglas Stebila

Joint work with Ross Evans
and Matthew McKague

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

NSERC CRSNG We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Primitives: KEM-DEM Security Pen & Paper Proof

CAPS 2025
https://prooffrog.github.io/caps-2025.html

Douglas Stebila

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

NSERC CRSNG We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Recap: Provable Security and Game Hopping Proofs

Recap of provable security

Main approach of reductionist security:

1. Define the syntax of the relevant primitives
2. Define security experiments for the relevant primitives
3. Specify your scheme
4. State a theorem bounding the success probability for a certain class of adversaries in breaking security of your scheme
   - usually depending on the success probability of breaking security of underlying primitives (and other terms)
5. Prove the theorem

Code-based game-playing proofs

- Papers
  - Shoup 2004
  - Bellare & Rogaway 2004
  - …

- Textbooks
  - Katz & Lindell, Introduction to Modern Cryptography
  - Rosulek, Joy of Cryptography
  - Boneh & Shoup, A Graduate Course in Applied Cryptography
  - …

CODE-BASED GAME-PLAYING PROOFS

A **security definition** is an experiment (expressed in pseudocode) with oracles

Different ways of structuring the experiment & oracles

1. main function that explicitly calls adversary with specified oracles

2. initialize / adversary access to all oracles / finalize

3. initialize / adversary access to all oracles + direct adversary output

Different experiment styles

1. **single-game win/lose**
   (secure if success probability ≈ 0)

2. **single-game hidden bit guessing**
   (secure if success probability ≈ ½)

3. **two-game indistinguishability:**
   left/right, real/random, real/ideal, ...
   (secure if distinguishing advantage ≈ 0)

Can even do this style for traditional win/lose games like unforgeability: e.g. "check" oracle that runs verify (real) versus rejects if not on a list (ideal)

Terminology

- A **library** is a collection of algorithms (each with input/output interfaces) and private variables that the algorithms can access.

- An algorithm can call into a library. The combined program is denoted $\mathcal{A} \diamond \mathcal{L}$

Libraries for security definitions

- We will use libraries to formalize a security definition:
  - private variables for the experiment
  - initialize routine
  - oracles that the adversary can call
- For a scheme Σ in a two-game indistinguishability experiment $(\mathcal{L}_{left}, \mathcal{L}_{right})$ we want to show that, for all programs $\mathcal{A}$

$$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{left} \Rightarrow \mathsf{true}] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{right} \Rightarrow \mathsf{true}]$$

Terminology

- **Inlining** library $\mathcal{L}$ into program (or library) $\mathcal{A}$: inserting the code from library $\mathcal{L}$ into another program $\mathcal{A}$ in every place where a subroutine of library $\mathcal{L}$ is called
- **Interchangeable:** libraries $\mathcal{L}_1$ and $\mathcal{L}_2$ are **interchangeable** (denoted $\mathcal{L}_1 \equiv \mathcal{L}_2$) if, for all programs $\mathcal{A}$ it holds that

$$\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow \mathsf{true}] = \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow \mathsf{true}]$$

  - Interchangeability comes up in "rewriting steps" in game-hopping proofs.)
  - One way of showing interchangeability is to show that $\mathcal{L}_1$ and $\mathcal{L}_2$ is to show that they are "code-wise equivalent", meaning they have the same source code, or "equivalent" source code
- **Indistinguishability**: libraries $\mathcal{L}_1$ and $\mathcal{L}_2$ are **indistinguishable** if, for all programs $\mathcal{A}$ it holds that

$$\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow \mathsf{true}] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow \mathsf{true}]$$

Game-hopping proofs

Goal: Show $\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{left} \Rightarrow \mathsf{true}] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{right} \Rightarrow \mathsf{true}]$

- Game 0 = $\mathcal{L}^{\Sigma}_{left}$ is the inlining of the code for your scheme Σ into security experiment $\mathcal{L}_{left}$
- Game 1 is another game (library) typically formed by changing some lines of Game 0
- Game 1 and Game 0 could be indistinguishable for one of several reasons:
  - They are in fact interchangeable (code-wise equivalent)
  - They are indistinguishable under some computational assumption
  - They are indistinguishable under some statistical argument

Game-hopping proofs

- Suppose we want to show Game 0 and Game 1 are indistinguishable under some computational assumption.
- Namely suppose scheme Γ satisfies a two-game security notion $(\mathcal{M}_{left}, \mathcal{M}_{right})$

$$\Pr[\mathcal{B} \diamond \mathcal{M}^{\Gamma}_{left} \Rightarrow \mathsf{true}] \approx \Pr[\mathcal{B} \diamond \mathcal{M}^{\Gamma}_{right} \Rightarrow \mathsf{true}]$$

- We define a reduction $\mathcal{R}$ that is an adversary to $(\mathcal{M}^{\Gamma}_{left}, \mathcal{M}^{\Gamma}_{right})$ such that

$$\mathcal{R} \diamond \mathcal{M}^{\Sigma}_{left} \equiv \mathrm{Game}_0$$

$$\mathcal{R} \diamond \mathcal{M}^{\Sigma}_{right} \equiv \mathrm{Game}_1$$

- We can conclude that Game 0 and Game 1 are indistinguishable assuming Γ is secure

Game-hopping proofs

Goal: Show $\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{left} \Rightarrow \mathsf{true}] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{right} \Rightarrow \mathsf{true}]$

- Repeat game hops as many times as needed until we arrive at a Game *n* such that

$$\text{Game}_n \equiv \mathcal{L}^{\Sigma}_{right}$$

To summarize, a proof consists of

1. Specifying each intermediate game (some can technically be omitted if they are implied by the reductions)
2. Justifying each game hop
   - If using indistinguishability:
     1. Specifying the reduction for each hop
     2. Justifying that each reduction inlined to its left/right game is code-wise equivalent to the previous/next game

KEM-DEM is IND-CPA

in the Joy of Cryptography style

with figures by Mike Rosulek

https://garbledcircus.com/kemdem/left-right

Goal: KEM-DEM is IND-CPA

Construction:
Build a public key encryption scheme by

- using a key encapsulation mechanism to compute a shared secret,
- and use that shared secret in a symmetric encryption scheme (data encapsulation mechanism) to encrypt a message.

Security:
Show that the KEM-DEM approach yields an IND-CPA-secure public key encryption scheme assuming that

- the KEM is IND-CPA-secure
- and the symmetric encryption scheme has one-time secrecy

Goal: KEM-DEM is IND-CPA

Idea of the proof:

- Game 0 = Starting game: Encrypt left message under real key
- Game 1: Use random KEM shared secret instead of real
- Game 2: Encrypt right message instead of left (under random key)
- Game 3: Use real KEM shared secret instead of random
  - Game 3 = Ending game: Encrypt right message under real key

If we want to be thorough, we need to:

1. **Symmetric encryption scheme:** define (a) syntax; (b) one-time secrecy
2. **Key encapsulation mechanism:** define (a) syntax; (b) IND-CPA security
3. **Public key encryption scheme:** define (a) syntax; (b) IND-CPA security
4. State the **KEM-DEM scheme**
5. Give a **game-hopping proof** for IND-CPA security of KEM-DEM
    1. State intermediate games (can be implicit)
    2. Give reductions to security of KEM or DEM
    3. Justify interchangeability / indistinguishability
6. State the **theorem** we just proved

Opinionated choices for this proof

- In the style of *Joy of Cryptography* by Mike Rosulek
- All security experiments are two-game indistinguishability: left/right, real/random
- All security experiments structured with initialize / adversary access to all oracles + direct adversary output
- Adversary gets setup values via oracles rather than direct input

1.a) Syntax of symmetric encryption scheme

A **symmetric-key encryption (SKE) scheme** consists of the following algorithms:

- Enc: a (possibly randomized) algorithm that takes a key $K \in \mathcal{K}$ and plaintext $M \in \mathcal{M}$ as input, and outputs a ciphertext $C \in \mathcal{C}$.
- Dec: a deterministic algorithm that takes a key $K \in \mathcal{K}$ and ciphertext $C \in \mathcal{C}$ as input, and outputs a plaintext $M \in \mathcal{M}$.

1.b) One-time secrecy of symmetric encryption

An encryption scheme Σ has **computational one-time secrecy (cOTS)** if the following two libraries are indistinguishable:

$\mathcal{L}^{\Sigma}_{\text{ske-ots-left}}$		$\mathcal{L}^{\Sigma}_{\text{ske-ots-right}}$
$\underline{\text{SKE.OTS.ENC}(M_L, M_R)\text{:}}$		$\underline{\text{SKE.OTS.ENC}(M_L, M_R)\text{:}}$
$K \twoheadleftarrow \Sigma.\mathcal{K}$	$\approxeq$	$K \twoheadleftarrow \Sigma.\mathcal{K}$
$C := \Sigma.\mathsf{Enc}(K, M_L)$		$C := \Sigma.\mathsf{Enc}(K, M_R)$
return C		return C

Note that $\mathcal{L}_{\text{ske-ots-rand}}$ makes no restriction about the lengths of M_L and M_R. Thus, the definition is suitable when all plaintexts have a known, fixed length.

2.a) Syntax of key encapsulation mechanism

A **key encapsulation mechanism** (KEM) consists of the following algorithms:

- KeyGen: same as in a PKE scheme, a randomized algorithm that takes no inputs and outputs a keypair (PK, SK).
- Encaps: a randomized algorithm that takes only a public key PK as input and returns both a ciphertext $C \in \mathcal{C}$ and plaintext $M \in \mathcal{M}$.
- Decaps: same as in a PKE scheme, a deterministic algorithm that takes a private key SK and ciphertext $C \in \mathcal{C}$ as input, and returns a plaintext $M \in \mathcal{M}$ (or raises an error).

2.b) IND-CPA security of a KEM

A KEM Σ has **security against chosen-plaintext attacks (CPA security)** if the following two libraries are indistinguishable:

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-real}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK( ):

  return PK

KEM.CPA.ENC( ):

  $(C, M) := \Sigma.\text{Encaps}(PK)$

  return (C, M)

$\approx$

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-ideal}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK( ):

  return PK

KEM.CPA.ENC( ):

  $(C, -) := \Sigma.\text{Encaps}(PK)$

  $M \twoheadleftarrow \Sigma.\mathcal{M}$

  return (C, M)

Initialize

Experiment private variables

Adversary gets public key via oracle

3.a) Syntax of public key encryption

A **public-key encryption** (PKE) scheme consists of the following algorithms:

- KeyGen: a randomized algorithm that takes no inputs (besides the security parameter, which we never write explicitly) and outputs a **key pair** (PK, SK), where PK is a **public key** and SK is a **private key**.
- Enc: a randomized algorithm that takes a public key PK and plaintext $M \in \mathcal{M}$ as input and returns a ciphertext $C \in \mathcal{C}$.
- Dec: a deterministic algorithm that takes a private key SK and ciphertext $C \in \mathcal{C}$ as input, and returns a plaintext $M \in \mathcal{M}$ (or raises an error).

3.b) IND-CPA security of a PKE

A PKE scheme Σ has **security against chosen-plaintext attacks (CPA security)** if the following two libraries are indistinguishable:

$\mathcal{L}^{\Sigma}_{\text{pke-cpa-left}}$		$\mathcal{L}^{\Sigma}_{\text{pke-cpa-right}}$
$(PK, SK) := \Sigma.\text{KeyGen}()$		$(PK, SK) := \Sigma.\text{KeyGen}()$
$\underline{\text{PKE.CPA.PK}(\,)}$:		$\underline{\text{PKE.CPA.PK}(\,)}$:
return PK	$\approx$	return PK
$\underline{\text{PKE.CPA.ENC}(M_L, M_R)}$:		$\underline{\text{PKE.CPA.ENC}(M_L, M_R)}$:
$C := \Sigma.\text{Enc}(PK, M_L)$		$C := \Sigma.\text{Enc}(PK, M_R)$
return C		return C

As above, the definition is suitable when plaintexts have a known, fixed length, since PKE.CPA.ENC of $\mathcal{L}_{\text{pke-cpa-rand}}$ does not restrict the lengths of M_L and M_R.

4. State the KEM-DEM scheme

Let KEM be a KEM scheme and DEM be a SKE scheme, such that $\mathsf{KEM}.\mathcal{M} = \mathsf{DEM}.\mathcal{K}$ (*i.e.*, KEM payloads can be interepreted as keys in DEM). Then **hybrid encryption** $\mathsf{Hyb} = \mathsf{Hyb}[\mathsf{KEM}, \mathsf{DEM}]$ is defined by the following algorithms:

$$\mathsf{Hyb}.\mathcal{K} = \mathsf{KEM}.\mathcal{K}$$
$$\mathsf{Hyb}.\mathcal{M} = \mathsf{DEM}.\mathcal{M}$$
$$\mathsf{Hyb}.\mathcal{C} = \mathsf{KEM}.\mathcal{C} \times \mathsf{DEM}.\mathcal{C}$$

$$\mathsf{Hyb}.\mathsf{KeyGen} = \mathsf{KEM}.\mathsf{KeyGen}$$

$\underline{\mathsf{Hyb}.\mathsf{Enc}(PK, M)}$:

$(C_{\text{kem}}, K) \twoheadleftarrow \mathsf{KEM}.\mathsf{Encaps}(PK)$

$C_{\text{dem}} \twoheadleftarrow \mathsf{DEM}.\mathsf{Enc}(K, M)$

return $(C_{\text{kem}}, C_{\text{dem}})$

$\underline{\mathsf{Hyb}.\mathsf{Dec}(SK, (C_{\text{kem}}, C_{\text{dem}}))}$:

$K := \mathsf{KEM}.\mathsf{Decaps}(SK, C_{\text{kem}})$

if $K == \bot$: return $\bot$

return $\mathsf{DEM}.\mathsf{Dec}(K, C_{\text{dem}})$

5. Proof: Game 0: Inline KEM-DEM scheme into CPA-left game

The starting point is $\mathcal{L}^{\mathsf{Hyb}}_{\mathsf{pke\text{-}cpa\text{-}left}}$.

$\mathcal{L}^{\mathsf{Hyb}}_{\mathsf{pke\text{-}cpa\text{-}left}}$

// Hyb.KeyGen():

$(PK, SK) :=$ KEM.KeyGen()

PKE.CPA.PK():

  return PK

PKE.CPA.ENC(M_L, M_R):

  // Hyb.Enc(PK, M_L):

  $(C_{\text{kem}}, K) \twoheadleftarrow$ KEM.Encaps(PK)

  $C_{\text{dem}} \twoheadleftarrow$ DEM.Enc(K, M_L)

  return $(C_{\text{kem}}, C_{\text{dem}})$

5. Proof: Game 0 is equivalent to a reduction calling into the CPA-real game for the KEM

Rewrite in a logically equivalent way so that an instance of $\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-real}}$ appears.

$PK := \text{KEM.CPA.PK}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, K) := \text{KEM.CPA.ENC}()$

  $C_{\text{dem}} \twoheadleftarrow \text{DEM.Enc}(K, M_L)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-real}}$

$(PK, SK) := \text{KEM.KeyGen}()$

$\underline{\text{KEM.CPA.PK}():}$

  return PK

$\underline{\text{KEM.CPA.ENC}():}$

  $(C, M) := \text{KEM.Encaps}(PK)$

  return (C, M)

[

5. Proof: Hop to Game 1 by switching the KEM CPA-real game to CPA-ideal

KEM is CPA-secure, so $\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-real}}$ can be replaced by $\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-ideal}}$ with only negligible effect on the calling program.

$PK := \text{KEM.CPA.PK}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, K) := \text{KEM.CPA.ENC}()$

  $C_{\text{dem}} \twoheadleftarrow \text{DEM.Enc}(K, M_L)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-ideal}}$

$(PK, SK) := \text{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{KEM.CPA.ENC}():}$

  $(C, -) := \text{KEM.Encaps}(PK)$

  $M \twoheadleftarrow \text{KEM}.\mathcal{M}$

  return (C, M)

[

5. Proof: Game 1: Write out Game 1 explicitly by inlining previous slide

Inline the instance of $\mathcal{L}^{\mathsf{KEM}}_{\mathsf{kem\text{-}cpa\text{-}ideal}}$.

$(PK, SK) := \mathsf{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, -) := \mathsf{KEM.Encaps}(PK)$

  $K \twoheadleftarrow \mathsf{KEM}.\mathcal{M}$

  $C_{\text{dem}} \twoheadleftarrow \mathsf{DEM.Enc}(K, M_L)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

[

5. Proof: Game 1: Note that KEM shared secret space equals symmetric key encryption space

By our assumption, $\mathsf{KEM}.\mathcal{M} = \mathsf{DEM}.\mathcal{K}$.

$(PK, SK) := \mathsf{KEM.KeyGen}()$

<u>PKE.CPA.PK():</u>

  return PK

<u>PKE.CPA.ENC(M_L, M_R):</u>

  $(C_{\text{kem}}, -) := \mathsf{KEM.Encaps}(PK)$

  $K \twoheadleftarrow \mathsf{DEM}.\mathcal{K}$

  $C_{\text{dem}} \twoheadleftarrow \mathsf{DEM.Enc}(K, M_L)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

[

5. Proof: Game 1 is equivalent to a reduction calling into the OTS-left game for the DEM

Rewrite in a logically eqivalent way, so that an instance of $\mathcal{L}^{\text{DEM}}_{\text{ske-ots-left}}$ appears.

$(PK, SK) := \text{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, -) := \text{KEM.Encaps}(PK)$

  $C_{\text{dem}} := \text{SKE.OTS.ENC}(M_L, M_R)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\text{DEM}}_{\text{ske-ots-left}}$

$\underline{\text{SKE.OTS.ENC}(M_L, M_R):}$

  $K \twoheadleftarrow \text{DEM}.\mathcal{K}$

  $C := \text{DEM.Enc}(K, M_L)$

  return C

[

5. Proof: Hop to Game 2 by switching the DEM OTS-left game to OTS-right

DEM has cOTS security, so $\mathcal{L}^{\mathsf{DEM}}_{\mathsf{ske\text{-}ots\text{-}left}}$ can be replaced by $\mathcal{L}^{\mathsf{DEM}}_{\mathsf{ske\text{-}ots\text{-}right}}$ with only negligible effect on the calling program.

$(PK, SK) :=$ KEM.KeyGen()
<u>PKE.CPA.PK():</u>
return PK
<u>PKE.CPA.ENC(M_L, M_R):</u>
$(C_{\text{kem}}, -) :=$ KEM.Encaps(PK)
$C_{\text{dem}} :=$ SKE.OTS.ENC(M_L, M_R)
return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\mathsf{DEM}}_{\mathsf{ske\text{-}ots\text{-}right}}$
<u>SKE.OTS.ENC(M_L, M_R):</u>
$K \leftarrow$ DEM.$\mathcal{K}$
$C :=$ DEM.Enc(K, M_R)
return C

[

5. Proof: Game 2: Write out Game 2 explicitly by inlining previous slide

Inline the instance of $\mathcal{L}^{\mathsf{DEM}}_{\mathsf{ske\text{-}ots\text{-}right}}$.

$(PK, SK) := \mathsf{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, -) := \mathsf{KEM.Encaps}(PK)$

  $K \twoheadleftarrow \mathsf{DEM}.\mathcal{K}$

  $C_{\text{dem}} := \mathsf{DEM.Enc}(K, M_R)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

Now we need to undo the use of a random encryption key

5. Proof: Game 2: Note that KEM shared secret space equals symmetric key encryption space

The next few steps are identical to some previous steps, but taken in reverse order.

$(PK, SK) := \mathsf{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, -) := \mathsf{KEM.Encaps}(PK)$

  $K \twoheadleftarrow \mathsf{DEM}.\mathcal{K}$

  $C_{\text{dem}} := \mathsf{DEM.Enc}(K, M_R)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

[

5. Proof: Game 2 is equivalent to a reduction calling into the CPA-ideal game for the KEM

$PK := \text{KEM.CPA.PK}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, K) := \text{KEM.CPA.ENC}()$

  $C_{\text{dem}} := \text{DEM.Enc}(K, M_R)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-ideal}}$

$(PK, SK) := \text{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{KEM.CPA.ENC}():}$

  $(C, -) := \text{KEM.Encaps}(PK)$

  $M \twoheadleftarrow \text{KEM}.\mathcal{M}$

  return (C, M)

[

5. Proof: Hop to Game 3 by switching the KEM CPA-ideal game to CPA-real

$PK :=$ KEM.CPA.PK()

PKE.CPA.PK():

return PK

PKE.CPA.ENC(M_L, M_R):

$(C_{\text{kem}}, K) :=$ KEM.CPA.ENC()

$C_{\text{dem}} :=$ DEM.Enc(K, M_R)

return $(C_{\text{kem}}, C_{\text{dem}})$

$\diamond$

$\mathcal{L}^{\text{KEM}}_{\text{kem-cpa-real}}$

$(PK, SK) :=$ KEM.KeyGen()

PKE.CPA.PK():

return PK

KEM.CPA.ENC():

$(C, M) :=$ KEM.Encaps(PK)

return (C, M)

[

5. Proof: Game 3: Write out Game 3 explicitly by inlining previous slide

$(PK, SK) :=$ KEM.KeyGen()

PKE.CPA.PK():

  return PK

PKE.CPA.ENC(M_L, M_R):

  $(C_{\text{kem}}, K) :=$ KEM.Encaps(PK)

  $C_{\text{dem}} :=$ DEM.Enc(K, M_R)

  return $(C_{\text{kem}}, C_{\text{dem}})$

[

5. Proof: Game 3 is equivalent to the inlining of the KEM-DEM scheme into the PKE CPA-right game

What remains is exactly $\mathcal{L}^{\mathsf{Hyb}}_{\mathsf{pke\text{-}cpa\text{-}right}}$.

$\mathcal{L}^{\mathsf{Hyb}}_{\mathsf{pke\text{-}cpa\text{-}right}}$

// Hyb.KeyGen():

$(PK, SK) :=$ KEM.KeyGen()

PKE.CPA.PK():

  return PK

PKE.CPA.ENC(M_L, M_R):

  // Hyb.Enc(PK, M_R):

  $(C_{\text{kem}}, K) :=$ KEM.Encaps(PK)

  $C_{\text{dem}} :=$ DEM.Enc(K, M_R)

  return $(C_{\text{kem}}, C_{\text{dem}})$

5. Proof summary

Game 0: KEM-DEM scheme in PKE CPA-left game

Game 1: Use random KEM shared secret instead of real

- Reduction R1 against KEM CPA security game
  - R1 with KEM-CPA-real = Game 0
  - R1 with KEM-CPA-ideal = Game 1

Game 2: Encrypt right message instead of left (under random key)

- Reduction R2 against DEM OTS security game
  - R2 with DEM-OTS-left = Game 1
  - R2 with DEM-OTS-right = Game 2

Game 3: Use real KEM shared secret instead of random

- Reduction R3 against KEM CPA security game
  - R3 with KEM-CPA-ideal = Game 2
  - R3 with KEM-CPA-real = Game 3
- Game 3 = KEM-DEM scheme in PKE-CPA-right game

6. Theorem statement

Theorem. Let KEM be a key encapsulation mechanism and DEM be a symmetric encryption scheme such that $\mathsf{KEM}.\mathcal{M} = \mathsf{DEM}.\mathcal{K}$. Let Hyb be the hybrid KEM-DEM scheme built from KEM and DEM. For every adversary $\mathcal{A}$, there exists reductions $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ (with small runtime) such that

$$\mathrm{Adv}^{\mathsf{CPA}}_{\mathsf{Hyb}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{R}_1^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{OTS}}_{\mathsf{DEM}}(\mathcal{R}_2^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{CPA}}_{\mathsf{KEM}}(\mathcal{R}_3^{\mathcal{A}})$$

KEM-DEM &more in ProofFrog

CAPS 2025
https://prooffrog.github.io/
https://eprint.iacr.org/2025/418

Douglas Stebila

Joint work with Ross Evans
and Matthew McKague

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

NSERC CRSNG
We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

ProofFrog

- A new tool for representing and checking cryptographic game-hopping proofs in the computational model
- Focus on accessibility & syntax for pen-and-paper cryptographers
- Limited in scope, strength, expressivity, & more compared to other tools

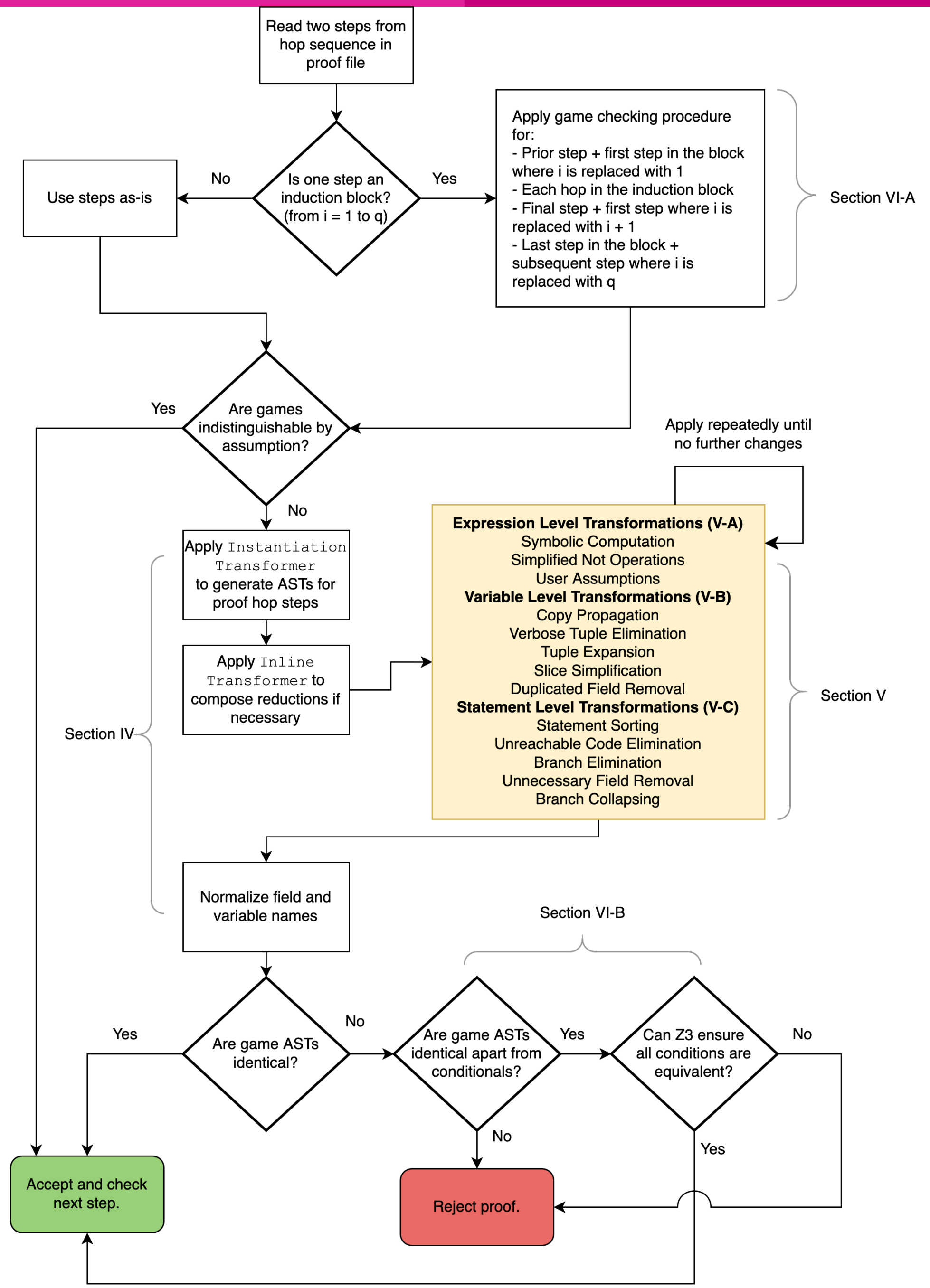
- Able to verify several Joy of Cryptography-style textbook examples
- Not (yet) suitable for richer research-level proofs

ProofFrog's approach

- The author of the proof states the reductions for each hop and optionally intermediate games.
- ProofFrog tries to evaluate the validity of each game hop by checking **code-wise equivalence** of each step
- Code-wide equivalence is checked by taking each game to be compared and applying a series of **automated transformations** to try to coerce the game into a "**canonical form**", and then comparing these canonical forms as strings
  - ProofFrog works with Abstract Syntax Trees
  - Examples of transformations: canonicalizing variable names, sorting sequence of statements, removing unused variables & statements, ...
- If ProofFrog's automated transformations manage to yield same canonical form: 😁
- If ProofFrog's automated transformations don't suffice: out of luck

ProofFrog engine

See https://eprint.iacr.org/2025/418 for details



KEM-DEM is IND-CPA

in ProofFrog

https://prooffrog.github.io/caps-2025.html

If we want to be thorough, we need to:

1. **Symmetric encryption scheme:** define (a) syntax; (b) one-time secrecy
2. **Key encapsulation mechanism:** define (a) syntax; (b) IND-CPA security
3. **Public key encryption scheme:** define (a) syntax; (b) IND-CPA security
4. State the **KEM-DEM scheme**
5. Give a **game-hopping proof** for IND-CPA security of KEM-DEM
    1. State intermediate games (can be implicit)
    2. Give reductions to security of KEM or DEM
    3. Justify interchangeability / indistinguishability
6. State the **theorem** we just proved

1.a) Syntax of symmetric encryption scheme

A **symmetric-key encryption (SKE) scheme** consists of the following algorithms:

- Enc: a (possibly randomized) algorithm that takes a key $K \in \mathcal{K}$ and plaintext $M \in \mathcal{M}$ as input, and outputs a ciphertext $C \in \mathcal{C}$.
- Dec: a deterministic algorithm that takes a key $K \in \mathcal{K}$ and ciphertext $C \in \mathcal{C}$ as input, and outputs a plaintext $M \in \mathcal{M}$.

Definition is parameterized by some sets

```
Primitive SymEnc(Set MessageSpace, Set CiphertextSpace, Set KeySpace) {
    Set Message = MessageSpace;
    Set Ciphertext = CiphertextSpace;
    Set Key = KeySpace;

    Ciphertext Enc(Key k, Message m);
    Message Dec(Key k, Ciphertext c);
}
```

Algorithm signatures

1.b) One-time secrecy of symmetric encryption

$\mathcal{L}^{\Sigma}_{\text{ske-ots-left}}$		$\mathcal{L}^{\Sigma}_{\text{ske-ots-right}}$
SKE.OTS.ENC(M_L, M_R): $K \twoheadleftarrow \Sigma.\mathcal{K}$ $C := \Sigma.\mathsf{Enc}(K, M_L)$ return C	$\approx$	SKE.OTS.ENC(M_L, M_R): $K \twoheadleftarrow \Sigma.\mathcal{K}$ $C := \Sigma.\mathsf{Enc}(K, M_R)$ return C

```
Game Left(SymEnc E) {
    E.Ciphertext ENC(E.Message mL, E.Message mR) {
        E.Key k <- E.Key;
        E.Ciphertext c = E.Enc(k, mL);
        return c;
    }
}
```

```
Game Right(SymEnc E) {
    E.Ciphertext ENC(E.Message mL, E.Message mR) {
        E.Key k <- E.Key;
        E.Ciphertext c = E.Enc(k, mR);
        return c;
    }
}
```

Observe that variables are typed

2.a) Syntax of key encapsulation mechanism

Definition is parameterized by some sets

```
Primitive KEM(Set SharedSecretSpace, Set CiphertextSpace, Set PKeySpace, Set SKeySpace) {
    Set SharedSecret = SharedSecretSpace;
    Set Ciphertext = CiphertextSpace;
    Set PublicKey = PKeySpace;
    Set SecretKey = SKeySpace;

    PublicKey * SecretKey KeyGen();
    Ciphertext * SharedSecret Encaps(PublicKey pk);
    SharedSecret Decaps(SecretKey sk, Ciphertext m);
}
```

Tuple

Algorithm signatures

2.b) IND-CPA security of a KEM

```
Game Real(KEM K) {
    K.PublicKey pk;
    K.SecretKey sk;

    Void Initialize() {
        K.PublicKey ∗ K.SecretKey k = K.KeyGen();
        pk = k[0];
        sk = k[1];
    }

    K.PublicKey PK() {
        return pk;
    }

    K.SharedSecret ∗ K.Ciphertext ENC() {
        K.Ciphertext ∗ K.SharedSecret rsp = K.Encaps(pk);
        return rsp;
    }
}
```

Experiment private variables

Initialize

Adversary gets public key via oracle

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-real}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK():

return PK

KEM.CPA.ENC():

$(C, M) := \Sigma.\text{Encaps}(PK)$

return (C, M)

$\approx$

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-ideal}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK():

return PK

KEM.CPA.ENC():

$(C, -) := \Sigma.\text{Encaps}(PK)$

$M \twoheadleftarrow \Sigma.\mathcal{M}$

return (C, M)

2.b) IND-CPA security of a KEM

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-real}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK( ):

return PK

KEM.CPA.ENC( ):

$(C, M) := \Sigma.\text{Encaps}(PK)$

return (C, M)

$\approx$

$\mathcal{L}^{\Sigma}_{\text{kem-cpa-ideal}}$

$(PK, SK) := \Sigma.\text{KeyGen}()$

KEM.CPA.PK( ):

return PK

KEM.CPA.ENC( ):

$(C, -) := \Sigma.\text{Encaps}(PK)$

$M \twoheadleftarrow \Sigma.\mathcal{M}$

return (C, M)

```
Game Ideal(KEM K) {
    K.PublicKey pk;
    K.SecretKey sk;

    Void Initialize() {
        K.PublicKey * K.SecretKey k = K.KeyGen();
        pk = k[0];
        sk = k[1];
    }

    K.PublicKey PK() {
        return pk;
    }

    K.SharedSecret * K.Ciphertext ENC() {
        K.Ciphertext * K.SharedSecret rsp = K.Encaps(pk);
        K.Ciphertext ctxt = rsp[0];
        K.SharedSecret ss <- K.SharedSecret;
        return [ctxt, ss];
    }
}
```

3.a) Syntax of public key encryption

```
Primitive PubKeyEnc(Set MessageSpace, Set CiphertextSpace, Set PKeySpace, Set SKeySpace) {
    Set Message = MessageSpace;
    Set Ciphertext = CiphertextSpace;
    Set PublicKey = PKeySpace;
    Set SecretKey = SKeySpace;

    PublicKey * SecretKey KeyGen();
    Ciphertext Enc(PublicKey pk, Message m);
    Message Dec(SecretKey sk, Ciphertext m);
}
```

3.b) IND-CPA security of a PKE

```
Game Left(PubKeyEnc E) {
    E.PublicKey pk;
    E.SecretKey sk;

    Void Initialize() {
        E.PublicKey * E.SecretKey k = E.KeyGen();
        pk = k[0];
        sk = k[1];
    }

    E.PublicKey PK() {
        return pk;
    }

    E.Ciphertext ENC(E.Message mL, E.Message mR) {
        return E.Enc(pk, mL);
    }
}
```

```
Game Right(PubKeyEnc E) {
    E.PublicKey pk;
    E.SecretKey sk;

    Void Initialize() {
        E.PublicKey * E.SecretKey k = E.KeyGen();
        pk = k[0];
        sk = k[1];
    }

    E.PublicKey PK() {
        return pk;
    }

    E.Ciphertext ENC(E.Message mL, E.Message mR) {
        return E.Enc(pk, mR);
    }
}
```

4. State the KEM-DEM scheme

such that $\text{KEM}.\mathcal{M} = \text{DEM}.\mathcal{K}$

$$\text{Hyb}.\mathcal{K} = \text{KEM}.\mathcal{K}$$
$$\text{Hyb}.\mathcal{M} = \text{DEM}.\mathcal{M}$$
$$\text{Hyb}.\mathcal{C} = \text{KEM}.\mathcal{C} \times \text{DEM}.\mathcal{C}$$

$$\text{Hyb.KeyGen} = \text{KEM.KeyGen}$$

$\underline{\text{Hyb.Enc}(PK, M)\text{:}}$

$(C_{\text{kem}}, K) \twoheadleftarrow \text{KEM.Encaps}(PK)$

$C_{\text{dem}} \twoheadleftarrow \text{DEM.Enc}(K, M)$

return $(C_{\text{kem}}, C_{\text{dem}})$

$\underline{\text{Hyb.Dec}(SK, (C_{\text{kem}}, C_{\text{dem}}))\text{:}}$

$K := \text{KEM.Decaps}(SK, C_{\text{kem}})$

if $K == \bot$: return $\bot$

return $\text{DEM.Dec}(K, C_{\text{dem}})$

```
Scheme Hyb(KEM K, SymEnc E) extends PubKeyEnc {
    requires K.SharedSecret subsets E.Key;

    Set PublicKey = K.PublicKey;
    Set SecretKey = K.SecretKey;
    Set Message = E.Message;
    Set Ciphertext = K.Ciphertext * E.Ciphertext;

    PublicKey * SecretKey KeyGen() {
        return K.KeyGen();
    }

    Ciphertext Enc(PublicKey pk, Message m) {
        K.Ciphertext * K.SharedSecret x = K.Encaps(pk);
        K.Ciphertext c_kem = x[0];
        E.Key k_dem = x[1];
        E.Ciphertext c_dem = E.Enc(k_dem, m);
        return [c_kem, c_dem];
    }

    Message Dec(SecretKey sk, Ciphertext c) {
        K.Ciphertext c_kem = c[0];
        E.Ciphertext c_dem = c[1];
        K.SharedSecret k_dem = K.Decaps(sk, c_kem);
        return E.Dec(k_dem, c_dem);
    }
}
```

* This ProofFrog modeling doesn't capture rejection for simplicity (but could)

5. Proof: Setting up the theorem statement

- First we list all the sets and primitives used in the theorem statement:

```
let:
    Set SymMessageSpace;
    Set KEMSharedSecretSpace;
    Set SymCiphertextSpace;
    Set KEMCiphertextSpace;

    Set PubKeySpace;
    Set SecretKeySpace;

    SymEnc E = SymEnc(SymMessageSpace, SymCiphertextSpace, KEMSharedSecretSpace);
    KEM K = KEM(KEMSharedSecretSpace, KEMCiphertextSpace, PubKeySpace, SecretKeySpace);
    Hyb H = Hyb(K, E);
```

Sets

Primitives

Target scheme

Notice the DEM secret key space is equal to the KEM shared secret space

5. Proof: Theorem statement

- Now we can state the security assumptions on the primitives:

```
assume:
    OTS(E);
    CPAKEM(K);
```

- And the goal of the theorem:

```
theorem:
    CPA(H);
```

5. Proof summary

Game 0: KEM-DEM scheme in PKE CPA-left game

Game 1: Use random KEM shared secret instead of real

- Reduction R1 against KEM CPA security game
  - R1 with KEM-CPA-real = Game 0
  - R1 with KEM-CPA-ideal = Game 1

Game 2: Encrypt right message instead of left (under random key)

- Reduction R2 against DEM OTS security game
  - R2 with DEM-OTS-left = Game 1
  - R2 with DEM-OTS-right = Game 2

Game 3: Use real KEM shared secret instead of random

- Reduction R3 against KEM CPA security game
  - R3 with KEM-CPA-ideal = Game 2
  - R3 with KEM-CPA-real = Game 3
- Game 3 = KEM-DEM scheme in PKE-CPA-right game

```
games:
    // Game 0
    CPA(H).Left;
    CPAKEM(K).Real compose R1(E, K, H);

    // Game 1
    CPAKEM(K).Ideal compose R1(E, K, H);
    OTS(E).Left compose R2(E, K, H);

    // Game 2
    OTS(E).Right compose R2(E, K, H);
    CPAKEM(K).Ideal compose R3(E, K, H);

    // Game 3
    CPAKEM(K).Real compose R3(E, K, H);
    CPA(H).Right;
```

* Minor simplification of ProofFrog notation: all game lines should have "`against CPA(H).Adversary`" at the end

5. Proof summary

```
games:
    // Game 0
    CPA(H).Left;
    CPAKEM(K).Real compose R1(E, K, H);

    // Game 1
    CPAKEM(K).Ideal compose R1(E, K, H);
    OTS(E).Left compose R2(E, K, H);

    // Game 2
    OTS(E).Right compose R2(E, K, H);
    CPAKEM(K).Ideal compose R3(E, K, H);

    // Game 3
    CPAKEM(K).Real compose R3(E, K, H);
    CPA(H).Right;
```

Code-wise equivalence steps:

ProofFrog checks that these steps are code-wise equivalent by

- inlining the scheme & reduction into the game
- canonicalizing each game
- comparing the programs as strings

* Minor simplification of ProofFrog notation: all game lines should have "`against CPA(H).Adversary`" at the end

5. Proof summary

```
games:
    // Game 0
    CPA(H).Left;
    CPAKEM(K).Real compose R1(E, K, H);

    // Game 1
    CPAKEM(K).Ideal compose R1(E, K, H);
    OTS(E).Left compose R2(E, K, H);

    // Game 2
    OTS(E).Right compose R2(E, K, H);
    CPAKEM(K).Ideal compose R3(E, K, H);

    // Game 3
    CPAKEM(K).Real compose R3(E, K, H);
    CPA(H).Right;
```

Indistinguishable by assumption steps:

ProofFrog checks that these steps are indistinguishable by an assumption in the theorem statement:

- `assume: CPAKEM(K)` implies `CPAKEM(K).Real ≈ CPAKEM(K).Ideal`
- `assume: OTS(E)` implies `OTS(E).Left ≈ OTS(E).Right`

* Minor simplification of ProofFrog notation: all game lines should have "`against CPA(H).Adversary`" at the end

5. Proof summary

```
games:
    // Game 0
    CPA(H).Left;
    CPAKEM(K).Real compose R1(E, K, H);

    // Game 1
    CPAKEM(K).Ideal compose R1(E, K, H);
    OTS(E).Left compose R2(E, K, H);

    // Game 2
    OTS(E).Right compose R2(E, K, H);
    CPAKEM(K).Ideal compose R3(E, K, H);

    // Game 3
    CPAKEM(K).Real compose R3(E, K, H);
    CPA(H).Right;
```

All that we have left to do is write out the three reductions R1, R2, R3

* Minor simplification of ProofFrog notation: all game lines should have "`against CPA(H).Adversary`" at the end

5. Proof: Reduction R1

$PK := \text{KEM.CPA.PK}()$

$\underline{\text{PKE.CPA.PK}():}$

  return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

  $(C_{\text{kem}}, K) := \text{KEM.CPA.ENC}()$

  $C_{\text{dem}} \twoheadleftarrow \text{DEM.Enc}(K, M_L)$

  return $(C_{\text{kem}}, C_{\text{dem}})$

```
Reduction R1(SymEnc E, KEM K, Hyb H) compose CPAKEM(K) {
    H.PublicKey PK() {
        return challenger.PK();
    }
    H.Ciphertext ENC(H.Message mL, H.Message mR) {
        K.Ciphertext * K.SharedSecret y = challenger.ENC();
        K.Ciphertext c_kem = y[0];
        K.SharedSecret k_dem = y[1];
        E.Ciphertext c_dem = E.Enc(k_dem, mL);
        return [c_kem, c_dem];
    }
}
```

5. Proof: Reduction R2

$(PK, SK) := \text{KEM.KeyGen}()$

$\underline{\text{PKE.CPA.PK}():}$

return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

$(C_{\text{kem}}, -) := \text{KEM.Encaps}(PK)$

$C_{\text{dem}} := \text{SKE.OTS.ENC}(M_L, M_R)$

return $(C_{\text{kem}}, C_{\text{dem}})$

```
Reduction R2(SymEnc E, KEM K, Hyb H) compose OTS(E) {
    K.PublicKey pk;
    K.SecretKey sk;
    Void Initialize() {
        K.PublicKey * K.SecretKey k = K.KeyGen();
        pk = k[0];
        sk = k[1];
    }
    H.PublicKey PK() {
        return pk;
    }

    H.Ciphertext ENC(H.Message mL, H.Message mR) {
        K.Ciphertext * K.SharedSecret x = K.Encaps(pk);
        K.Ciphertext c_kem = x[0];
        E.Ciphertext c_dem = challenger.ENC(mL, mR);
        return [c_kem, c_dem];
    }
}
```

5. Proof: Reduction R3

$PK := \text{KEM.CPA.PK}()$

$\underline{\text{PKE.CPA.PK}():}$

return PK

$\underline{\text{PKE.CPA.ENC}(M_L, M_R):}$

$(C_{\text{kem}}, K) := \text{KEM.CPA.ENC}()$

$C_{\text{dem}} := \text{DEM.Enc}(K, M_R)$

return $(C_{\text{kem}}, C_{\text{dem}})$

```
Reduction R3(SymEnc E, KEM K, Hyb H) compose CPAKEM(K) {
    H.PublicKey PK() {
        return challenger.PK();
    }
    H.Ciphertext ENC(H.Message mL, H.Message mR) {
        K.Ciphertext * K.SharedSecret y = challenger.ENC();
        K.Ciphertext c_kem = y[0];
        K.SharedSecret k_dem = y[1];
        E.Ciphertext c_dem = E.Enc(k_dem, mR);
        return [c_kem, c_dem];
    }
}
```

We're done!

```
> proof_frog prove Hyb-is-CPA.proof

==STEP 1===
Current: CPA(H).Left;
Hop To: Game0(K, E, H);

SIMPLIFYING CURRENT GAME
Game Left() {

[…]

Inline Success!
Proof Succeeded!
```

- 3 files for primitive syntax: 27 LoC
- 3 files for security definitions: 83 LoC
- 1 file for scheme: 26 LoC
- 1 file for proof: 75 LoC

- Took me about 30 minutes to write it

Other examples from Joy of Cryptography in ProofFrog

Primitives and Associated Security Definitions.

- Symmetric Encryption Schemes [14, Definition 2.1]
  - Correctness [14, Definition 2.2]
  - One-Time Uniform Ciphertexts [14, Definition 2.5]
  - One-Time Secrecy [14, Definition 2.6]
  - CPA-security [14, Definition 7.1]
  - CPA$-security [14, Definition 7.2]
  - CCA-security [14, Definition 9.1]
  - CPA$-security [14, Definition 9.2]
- Pseudorandom Generators (PRGs) and security [14, Definition 5.1]
- Pseudorandom Functions (PRFs) and security [14, Definition 6.1]
- Message Authentication Codes (MACs) [14, Definition 10.1] and security [14, Definition 10.2]
- Public Key Encryption Schemes [14, Chapter 15]
  - Correctness [14, Chapter 15]
  - One-Time Secrecy [14, Definition 15.4]
  - CPA-security [14, Definition 15.1]
  - CPA$-security [14, Definition 15.2]

Completed Proofs.

- A symmetric encryption scheme that encrypts twice with a one-time-pad using independent keys has one-time uniform ciphertexts. [14, Claim 2.13].
- If a symmetric encryption scheme has one-time uniform ciphertexts, then it has one-time secrecy. [14, Theorem 2.15]
- If a symmetric encryption scheme Σ has one-time secrecy, then a symmetric encryption scheme which encrypts by returning a pair of ciphertexts (c_1, c_2) where $c_i = \Sigma.\text{Enc}(k_i, m)$ also has one-time secrecy. [14, Exercise 2.13]
- A symmetric encryption scheme Σ has one-time secrecy if and only if an encryption of a provided message with a one-time key is indistinguishable from an encryption of a random message with a one-time key. [14, Exercise 2.14]
- A symmetric encryption scheme Σ has one-time secrecy if and only if the ciphertext pair (c_L, c_R) is indistinguishable from the ciphertext (c_R, c_L) where m_L and m_R are encrypted with one-time keys. [14, Exercise 2.15]
- The Pseudo-OTP symmetric encryption scheme which uses a secure pseudo-random generator G to encrypt messages as $G(k) \oplus m$ provides one-time secrecy. [14, Claim 5.4]
- A length-tripling PRG which, when given a seed s, uses a length-doubling PRG G to compute $x \parallel y = G(s)$, $u \parallel v = G(y)$ and returns $x \parallel u \parallel v$ is secure assuming G's security. [14, Claim 5.5]
- Given a length-tripling PRG G, a PRG H which, when given a seed s, computes $x \parallel y \parallel z = G(s)$ and returns $G(x) \parallel G(z)$ is secure. [14, Exercise 5.8.a]
- Given a length-tripling PRG G, a PRG H which, when given a seed s, computes $x \parallel y \parallel z = G(s)$ and returns $x \parallel y$ is secure. [14, Exercise 5.8.b]
- Given a length-tripling PRG G, a PRG H which, when given a seed s, computes $x = G(s)$, $y = G(0^\lambda)$ and returns $x \oplus y$ is secure. [14, Exercise 5.8.e]
- Given a length-tripling PRG G, a PRG H which, when given a seed $s_L \parallel s_R$, computes $x = G(s_L)$, $y = G(s_R)$ and returns $x \oplus y$ is secure. [14, Exercise 5.8.f]
- Given a length-doubling PRG G, a PRG H which, when given a seed s, computes $x \parallel y = G(s)$, $w = G(y)$ and returns $(x \oplus y) \parallel w$ is secure. [14, Exercise 5.10]
- If a symmetric encryption scheme is CPA$-secure, then it is also CPA-secure. [14, Claim 7.3]
- A symmetric encryption scheme has CPA security if and only if encryptions of provided messages using the same key are indistinguishable from encryptions of random messages using the same key. [14, Exercise 7.13]
- If a symmetric encryption scheme is CCA$-secure, then it is also CCA-secure. [14, Exercise 9.6]
- If Σ is a CPA-secure symmetric encryption scheme and M is a secure MAC, then the encrypt-then-MAC construction is CCA-secure. [14, Claim 10.10]
- If a public-key encryption scheme has one-time secrecy, then it is also CPA-secure. [14, Claim 15.5]
- If Σ_{sym} is a one-time-secret symmetric-key encryption scheme and Σ_{pub} is a CPA-secure, then hybrid encryption which generates a one-time symmetric key, encrypts the symmetric key under Σ_{pub}, encrypts the message under the one-time symmetric key, and returns the pair of ciphertexts is a CPA-secure public-key encryption scheme. [14, Claim 15.9]
- If Σ_S and Σ_T are symmetric encryption schemes, where Σ_T has one-time uniform ciphertexts, then the encryption scheme Σ which encrypts a message first with Σ_S, and then encrypts the resulting ciphertext with Σ_T, also has one-time uniform ciphertexts.
- If Σ_S and Σ_T are symmetric encryption schemes, where Σ_T is CPA$-secure, then the encryption scheme Σ which encrypts a message first with Σ_S, and then encrypts the resulting ciphertext with Σ_T, is also CPA$-secure.

Neato: Variable-length hybrid argument in ProofFrog

```
games:

    CPA(E).Left;

    induction(i from 1 to q) {
        OneTimeSecrecy(E).Left compose R(E, i);
        OneTimeSecrecy(E).Right compose R(E, i);
    }

    CPA(E).Right;
```

* A few details omitted for presentation clarity

ProofFrog has many limitations

- No formal-verified base or precise semantics
- Very tied to the game-hopping formalism
- Restricted domain specific language
- Very little understanding of mathematics
- No manual intervention if proof engine fails
- No attempt yet at protocols with complex states
- Minimal tooling and documentation
- Minimal developer community

Q: What is the future of ProofFrog?

A: Uncertain; looking for feedback

Continue developing ProofFrog as a formal verification engine?

- Improve expressivity
- Option to fork out to EasyCrypt when stuck
- Export to LaTeX

Transition to a tool to support pen-and-paper cryptographers?

- Manage game source code for pen-and-paper proofs in a domai—specific language
- Some type-checking and minimal validation
- Export to LaTeX

Want to get started with ProofFrog?

- Easy to install with Python (`pip3 install proof_frog`)
- Engine and examples at https://github.com/ProofFrog/
- (Hopefully) fun way to write your first formally verified proof!
- Be aware of limitations
- Ask questions on Github Discussions
- Contact me (dstebila@uwaterloo.ca) if you have thoughts on the possible directions (formal verification engine? pen-and-paper support tool?) and want to help out

1) Primitives: KEM-DEM Security Pen & Paper Proof
2) KEM-DEM & more in ProofFrog

CAPS 2025
https://prooffrog.github.io/
https://eprint.iacr.org/2025/418

Douglas Stebila

Joint work with Ross Evans
and Matthew McKague

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

NSERC CRSNG We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).