

# Two proof tools for pen-and-paper cryptographers

**Douglas Stebila**



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

# SKECH 2014

## Provable security of advanced properties of TLS and SSH

Douglas Stebila

joint work with Ben Dowling (QUT),  
Florian Bergsma (né Giesen), Florian Kohlar,  
Jörg Schwenk (Bochum)

eprint 2012/630 (CCS'13), eprint 2013/813

2014/06/03

Secure key exchange and  
channels workshop  
Bertinoro, Italy



Supported by:

Australian Technology  
Network-German Academic  
Exchange Service (ATN-  
DAAD) Joint Research  
Cooperation Scheme

Australian Research Council  
Discovery Project

## Questions

- Should we be trying to cryptographically analyze these more complex properties?
- Is the monolithic ACCE framework the right approach?

# SKECH 2016

## On writing key exchange protocols

Douglas Stebila  McMaster University

### Why I'd like to use EasyCrypt

- Proofs are hard.
- Want to make sure my proofs are correct.
  - Sometimes I write out the games in detail.
  - Sometimes it's a bit more hand-wavy.

The more complex the security model,  
**the more hand-wavy my proofs tend to be.**

*but*

The more complex the security model,  
**the more rigorous my proofs should be.**

# SKECH 2016

## On writing key exchange protocols

---

Douglas Stebila  McMaster University

SKECH II • July 5, 2016

### Questions about pseudocode definitions

- Have you tried this before?
  - If so, why did you not use it? (or if you did use it, what was your experience?)
- Would you want to see this in a paper that you referee?
- Would you want to see full sequence-of-games proofs with explicit reductions in a paper that you referee?
- Shut up and stop navel gazing?

# **SKECH 2018**

# SKECH 2022

## Proving KEMTLS in Tamarin or: I used Tamarin and you can too!

**Douglas Stebila**



Based on joint work with Sofia Celi, Jonathan Hoyland, Thom Wiggers

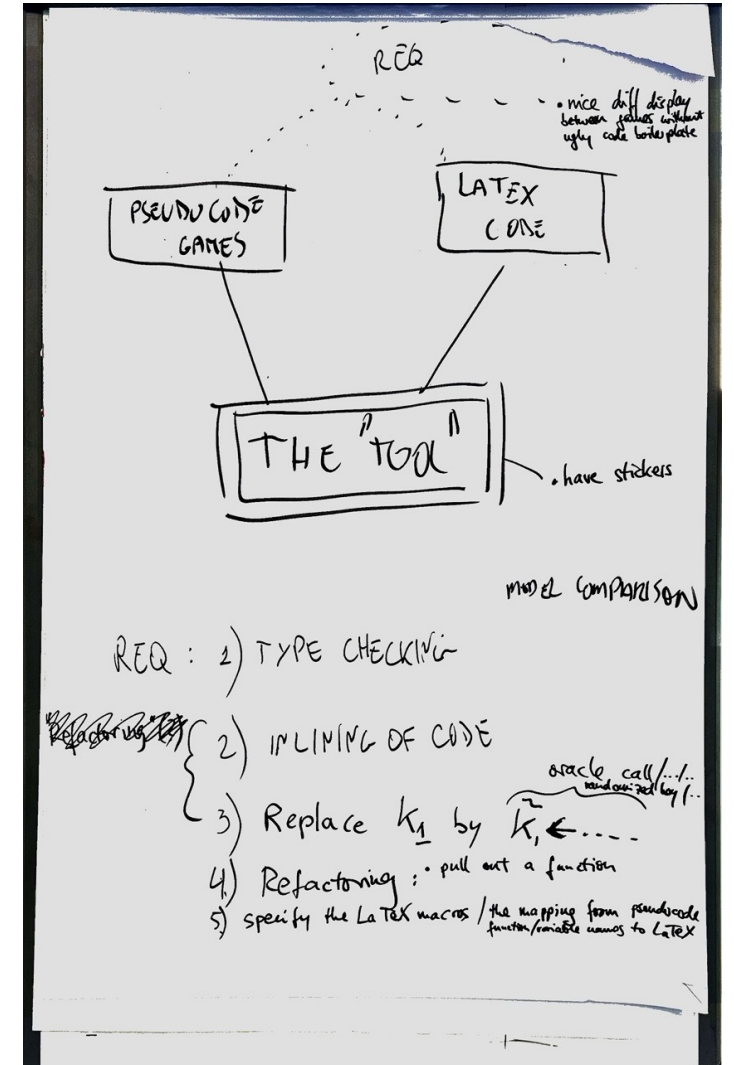
SKECH 4 • 2022-07-12

## Limitations of pen-and-paper proofs

- Mostly written out for session-key indistinguishability for KEMTLS and KEMTLS-PDK server-only auth variants
- No explicit games / reductions\*
- But only as reliable as the authors and the readers are
- Proof sketches for session-key indistinguishability of remaining variants
- Hand-waving argument for offline deniability
- Variants handled independently


In the public versions. I did write them out for KEMTLS in a private version with painful LaTeX macros.

# SKECH 2024



# Hierarchy of tool capabilities

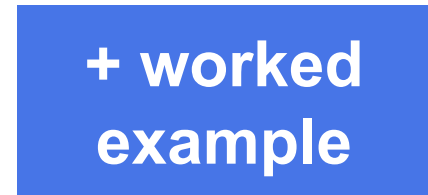
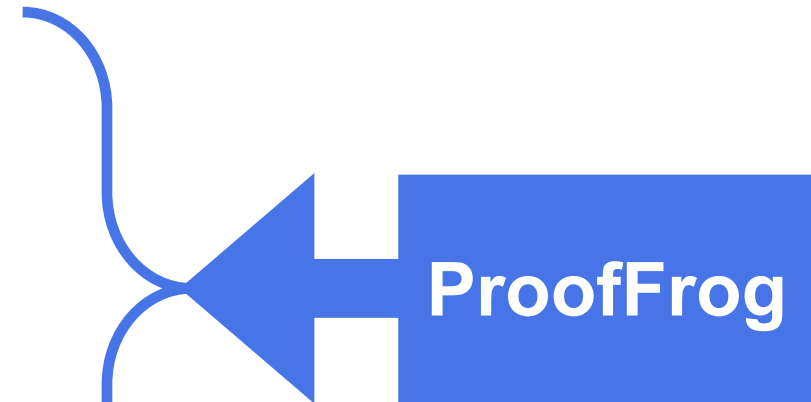
1. LaTeX management
2. Basic inlining & typechecking
3. More advanced typechecking
4. Exportable to formal verification



Require domain-specific language capable of expressing some/all of cryptography (interactive stateful oracle programs, rich mathematical types)

# Hierarchy of tool capabilities

1. LaTeX management
2. Basic inlining & typechecking
3. More advanced typechecking
4. Exportable to formal verification



# TeXFrog

<https://texfrog.github.io/>



# My goals

1. Help proof authors manage the LaTeX source code for their game-hopping proofs
  - Write a single block of LaTeX source code
  - Generate individual games, reductions, and consolidated game figures
2. Think about new ways of presenting proofs to readers

# Working in LaTeX

1. Tag lines of source code with metadata indicating which games/reductions a line belongs to
2. Generate figures with the relevant lines included

# 1. Tag lines of source code with metadata indicating which games/reductions a line belongs to

```
1 \begin{tfsource}{indcpa}
2 \begin{pcvstack}[boxed]
3   \procedure[linenumbering]{%
4     \tfontly*{G0}{Game $\tfgamename{G0} = \INDCPA_\Enc^{\Adversary()}$}%
5     \tfontly*{G1}{Game $\tfgamename{G1}$}%
6     \tfontly*{Red1} Reduction $\Bdversary_1^{\OPRF}$}%
7   }{
8     \tfontly{G0}{k \getsr \{0,1\}^\lambda \ \}
9     b \getsr \{0,1\} \ \}
10    b' \gets \Adversary^{\mathsf{LR}}() \ \}
11    \pcreturn (b' = b)
12  }
13  \pclb
14  \procedure[linenumbering]{$\mathsf{LR}(m_0, m_1)$}{
15    r \getsr \{0,1\}^\lambda \ \}
16    \tfontly{G0}{y \gets \mathrm{PRF}(k, r) \ \}
17    \tfontly{G1}{y \gets \RF(r) \ \}
18    \tfontly{Red1}{y \gets \OPRF(r) \ \}
19    \tfontly{G0-G2}{c \gets y \oplus m_b \ \}
20    \pcreturn (r, c)
21  }
22 \end{pcvstack}
23 \end{tfsource}
```

## 2. Generate figures with the relevant lines included

```
\tfreendgame[diff=G1]{indcpa}{G0}  
\tfreendgame[diff=G0]{indcpa}{G1}
```

Game  $G_0 = \text{IND-CPA}_{\text{Enc}}^{\mathcal{A}}()$

---

- 1 :  $k \leftarrow_{\$} \{0, 1\}^{\lambda}$
- 2 :  $b \leftarrow_{\$} \{0, 1\}$
- 3 :  $b' \leftarrow \mathcal{A}^{\text{LR}}()$
- 4 : **return**  $(b' = b)$

$\text{LR}(m_0, m_1)$

---

- 1 :  $r \leftarrow_{\$} \{0, 1\}^{\lambda}$
- 2 :  $y \leftarrow \text{PRF}(k, r)$
- 3 :  $c \leftarrow y \oplus m_b$
- 4 : **return**  $(r, c)$

Game  $G_1$

---

- 1 :  $b \leftarrow_{\$} \{0, 1\}$
- 2 :  $b' \leftarrow \mathcal{A}^{\text{LR}}()$
- 3 : **return**  $(b' = b)$

$\text{LR}(m_0, m_1)$

---

- 1 :  $r \leftarrow_{\$} \{0, 1\}^{\lambda}$
- 2 :  $y \leftarrow \text{RF}(r)$
- 3 :  $c \leftarrow y \oplus m_b$
- 4 : **return**  $(r, c)$

# New ways of presenting proofs to readers

**GAMES**

$G_0$   
IND-CPA $_{\text{Enc}}^A$ .Real(): the real IND-CPA game. The LR oracle encrypts using the actual PRF.

$G_1$   
Replace PRF( $k, r$ ) with a truly random value.

$\mathcal{B}_1$   
Reduction against PRF security, simulating Games  $G_0$  and  $G_1$  via an external PRF challenger.

$G_2$   
IND-CPA $_{\text{Enc}}^A$ .Ideal(): the ideal IND-CPA game. Ciphertext is uniformly random, independent of  $m_b$ .

← Prev

$G_1$

Replace PRF( $k, r$ ) with a truly random value.

Next →

$G_0$

Game  $G_0 = \text{IND-CPA}_{\text{Enc}}^A.\text{Real}()$

- 1:  $k \leftarrow_{\$} \{0, 1\}^\lambda$
- 2:  $b \leftarrow_{\$} \{0, 1\}$
- 3:  $b' \leftarrow \mathcal{A}^{\text{LR}}()$
- 4: **return** ( $b' = b$ )

---

LR( $m_0, m_1$ )

- 1:  $r \leftarrow_{\$} \{0, 1\}^\lambda$
- 2:  $y \leftarrow \text{PRF}(k, r)$
- 3:  $c \leftarrow y \oplus m_b$
- 4: **return** ( $r, c$ )

$G_1$

Game  $G_1$

- 1:  $k \leftarrow_{\$} \{0, 1\}^\lambda$
- 2:  $b \leftarrow_{\$} \{0, 1\}$
- 3:  $b' \leftarrow \mathcal{A}^{\text{LR}}()$
- 4: **return** ( $b' = b$ )

---

LR( $m_0, m_1$ )

- 1:  $r \leftarrow_{\$} \{0, 1\}^\lambda$
- 2:  $y \leftarrow_{\$} \{0, 1\}^\lambda$
- 3:  $c \leftarrow y \oplus m_b$
- 4: **return** ( $r, c$ )

**Claim.** Games  $G_0$  and  $G_1$  are computationally indistinguishable assuming PRF is a secure pseudorandom function, with advantage gap at most  $\text{Adv}^{\text{PRF}}(\mathcal{B}_1)$ .  
Reduction  $\mathcal{B}_1$  (shown separately) interacts with an external PRF challenger and simulates the LR oracle by forwarding nonces  $r$  to that challenger. When the external challenger uses the real PRF,  $\mathcal{B}_1$  exactly simulates  $G_0$ ; when it uses a random function,  $\mathcal{B}_1$  exactly simulates  $G_1$ .

− 100% +?

# A key exchange model

← Prev

$\mathcal{B}_{1,b}$

Next →

Reduction against  $q_{\text{gen}}$ -SUF-CMA security of SIG, simulating Games~1 and~2.

$\mathcal{G}_1^b$

**Game  $\mathcal{G}_1^b$**

```

00  $(n, m) \leftarrow 0^2$ 
01  $(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$ 
02  $(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$ 
03  $(\text{bad}_1, \text{bad}_2) \leftarrow 0^2$ 
04  $b' \leftarrow_{\mathcal{S}} \mathcal{A}$ 
05 Stop with  $b'$ 

Oracle Init( $pk$ )
06  $n \leftarrow n + 1$ 
07  $x \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
08  $c \leftarrow g^x$ 
09  $st_n \leftarrow (pk, x, c)$ 
10 If  $\exists j \in [m] : pk = pk_j$ :
11    $P[n] \leftarrow j; I[n] \leftarrow c$ 
12 Return  $c$ 

Oracle Receive( $i, c', ch$ )
13 Require  $i \in [n] \setminus Q$ 
14  $Q \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
15 If  $c' \in R[P[i], i]$ : Return
16  $(pk, x, c) \leftarrow st_i$ 
17  $(h, \sigma) \leftarrow c'$ 
18 If  $\text{SIG.vfy}(pk, (c, h), \sigma) = 0$ :
19   Return  $\perp$ 
20 Else:
21    $k \leftarrow H(c, h, h^x)$ 
22 If  $ch \wedge i \notin XP \wedge P[i] \notin CR$ :
23   If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
24    $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
25 Return  $k$ 

```

**Oracle Gen**

```

26  $m \leftarrow m + 1$ 
27  $(sk_m, pk_m) \leftarrow_{\mathcal{S}} \text{SIG.gen}$ 
28 If  $\exists i \in [n] : st_i = (pk_m, \cdot, \cdot) \wedge P[i] = \perp$ :
29    $\text{bad}_1 \leftarrow 1$ ; Stop
30 Return  $pk_m$ 

```

**Oracle Respond**( $j, c, ch$ )

```

31 Require  $j \in [m]$ 
32  $y \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
33  $h \leftarrow g^y$ 
34  $\sigma \leftarrow_{\mathcal{S}} \text{SIG.sig}(sk_j, (c, h))$ 
35  $c' \leftarrow (h, \sigma)$ 
36  $k \leftarrow H(c, h, c^y)$ 
37 If  $\exists i \in [n] : P[i] = j \wedge I[i] = c$ :
38    $R[j, i] \stackrel{\leftarrow}{\leftarrow} \{c'\}$ 
39   If  $ch \wedge i \notin XP$ :
40     If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
41      $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
42 Return  $(k, c')$ 

```

**Oracle Corrupt**( $j$ )

```

43 Require  $j \in [m]$ 
44  $CR \stackrel{\leftarrow}{\leftarrow} \{j\}$ 
45 Return  $sk_j$ 

```

**Oracle Expose**( $i$ )

```

46 Require  $i \in [n] \setminus ICH$ 
47  $XP \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
48 Return  $st_i$ 

```

**Random Oracle**  $H(X, Y, Z)$

```

49 If  $H[X, Y, Z] \neq \perp$ : Return  $H[X, Y, Z]$ 
50  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
51  $H[X, Y, Z] \leftarrow k$ 
52 Return  $k$ 

```

$\mathcal{B}_{1,b}$

**Adversary  $\mathcal{B}_{1,b}^{\text{Gen}, \text{Sign}, \text{Corrupt}'}$**

```

00  $(n, m) \leftarrow 0^2$ 
01  $(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$ 
02  $(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$ 
03  $b' \leftarrow_{\mathcal{S}} \mathcal{A}$ 
04 Stop

Oracle Init( $pk$ )
05  $n \leftarrow n + 1$ 
06  $x \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
07  $c \leftarrow g^x$ 
08  $st_n \leftarrow (pk, x, c)$ 
09 If  $\exists j \in [m] : pk = pk_j$ :
10    $P[n] \leftarrow j; I[n] \leftarrow c$ 
11 Return  $c$ 

```

**Oracle Receive**( $i, c', ch$ )

```

12 Require  $i \in [n] \setminus Q$ 
13  $Q \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
14 If  $c' \in R[P[i], i]$ : Return
15  $(pk, x, c) \leftarrow st_i$ 
16  $(h, \sigma) \leftarrow c'$ 
17 If  $\text{SIG.vfy}(pk, (c, h), \sigma) = 0$ :
18   Return  $\perp$ 
19 Else:
20    $k \leftarrow H(c, h, h^x)$ 
21 If  $ch \wedge i \notin XP \wedge P[i] \notin CR$ :
22   Stop with  $(P[i], (c, h), \sigma)$ 
23   If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
24    $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
25 Return  $k$ 

```

**Oracle Gen**

```

26  $m \leftarrow m + 1$ 
27  $pk_m \leftarrow \text{Gen}'$ 
28 If  $\exists i \in [n] : st_i = (pk_m, \cdot, \cdot) \wedge P[i] = \perp$ :
29   Stop
30 Return  $pk_m$ 

```

**Oracle Respond**( $j, c, ch$ )

```

31 Require  $j \in [m]$ 
32  $y \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
33  $\sigma \leftarrow \text{Sign}(j, (c, g^y))$ 
34  $k \leftarrow H(c, g^y, c^y)$ 
35  $c' \leftarrow (g^y, \sigma)$ 
36 If  $\exists i \in [n] : P[i] = j \wedge I[i] = c$ :
37    $R[j, i] \stackrel{\leftarrow}{\leftarrow} \{c'\}$ 
38   If  $ch \wedge i \notin XP$ :
39     If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
40      $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
41 Return  $(k, c')$ 

```

**Oracle Corrupt**( $j$ )

```

42 Require  $j \in [m]$ 
43  $sk_j \leftarrow \text{Corrupt}'(j)$ 
44  $CR \stackrel{\leftarrow}{\leftarrow} \{j\}$ 
45 Return  $sk_j$ 

```

**Oracle Expose**( $i$ )

```

46 Require  $i \in [n] \setminus ICH$ 
47  $XP \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
48 Return  $st_i$ 

```

**Random Oracle**  $H(X, Y, Z)$

```

49 If  $H[X, Y, Z] \neq \perp$ : Return
 $H[X, Y, Z]$ 
50  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
51  $H[X, Y, Z] \leftarrow k$ 
52 Return  $k$ 

```

$\mathcal{G}_2^b$

**Game  $\mathcal{G}_2^b$**

```

00  $(n, m) \leftarrow 0^2$ 
01  $(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$ 
02  $(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$ 
03  $(\text{bad}_1, \text{bad}_2) \leftarrow 0^2$ 
04  $b' \leftarrow_{\mathcal{S}} \mathcal{A}$ 
05 Stop with  $b'$ 

Oracle Init( $pk$ )
06  $n \leftarrow n + 1$ 
07  $x \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
08  $c \leftarrow g^x$ 
09  $st_n \leftarrow (pk, x, c)$ 
10 If  $\exists j \in [m] : pk = pk_j$ :
11    $P[n] \leftarrow j; I[n] \leftarrow c$ 
12 Return  $c$ 

```

**Oracle Receive**( $i, c', ch$ )

```

13 Require  $i \in [n] \setminus Q$ 
14  $Q \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
15 If  $c' \in R[P[i], i]$ : Return
16  $(pk, x, c) \leftarrow st_i$ 
17  $(h, \sigma) \leftarrow c'$ 
18 If  $\text{SIG.vfy}(pk, (c, h), \sigma) = 0$ :
19   Return  $\perp$ 
20 Else:
21    $k \leftarrow H(c, h, h^x)$ 
22 If  $ch \wedge i \notin XP \wedge P[i] \notin CR$ :
23    $\text{bad}_2 \leftarrow 1$ ; Stop
24   If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
25    $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
26 Return  $k$ 

```

**Oracle Gen**

```

27  $m \leftarrow m + 1$ 
28  $(sk_m, pk_m) \leftarrow_{\mathcal{S}} \text{SIG.gen}$ 
29 If  $\exists i \in [n] : st_i = (pk_m, \cdot, \cdot) \wedge P[i] = \perp$ :
30    $\text{bad}_1 \leftarrow 1$ ; Stop
31 Return  $pk_m$ 

```

**Oracle Respond**( $j, c, ch$ )

```

32 Require  $j \in [m]$ 
33  $y \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ 
34  $h \leftarrow g^y$ 
35  $\sigma \leftarrow_{\mathcal{S}} \text{SIG.sig}(sk_j, (c, h))$ 
36  $c' \leftarrow (h, \sigma)$ 
37  $k \leftarrow H(c, h, c^y)$ 
38 If  $\exists i \in [n] : P[i] = j \wedge I[i] = c$ :
39    $R[j, i] \stackrel{\leftarrow}{\leftarrow} \{c'\}$ 
40   If  $ch \wedge i \notin XP$ :
41     If  $b = 1$ :  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
42      $ICH \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
43 Return  $(k, c')$ 

```

**Oracle Corrupt**( $j$ )

```

44 Require  $j \in [m]$ 
45  $CR \stackrel{\leftarrow}{\leftarrow} \{j\}$ 
46 Return  $sk_j$ 

```

**Oracle Expose**( $i$ )

```

47 Require  $i \in [n] \setminus ICH$ 
48  $XP \stackrel{\leftarrow}{\leftarrow} \{i\}$ 
49 Return  $st_i$ 

```

**Random Oracle**  $H(X, Y, Z)$

```

50 If  $H[X, Y, Z] \neq \perp$ : Return  $H[X, Y, Z]$ 
51  $k \leftarrow_{\mathcal{S}} \mathcal{K}$ 
52  $H[X, Y, Z] \leftarrow k$ 
53 Return  $k$ 

```

# A starting point, not an ending point

## ⚠ Caveat: ⚠

I haven't actually written a paper using TeXFrog, so I don't know if it actually helps proof authors

## Call for discussion:

- What is useful to us as proof authors?
- What would we like as proof readers?
- With coding agents, the cost of making tools like this can be low.\*

\* TeXFrog was entirely vibe-coded with Claude Code in about 10 hours

# ProofFrog

<https://prooffrog.github.io/>



# Game hopping proofs, Joy of Cryptography-style

All security properties defined as left/right games

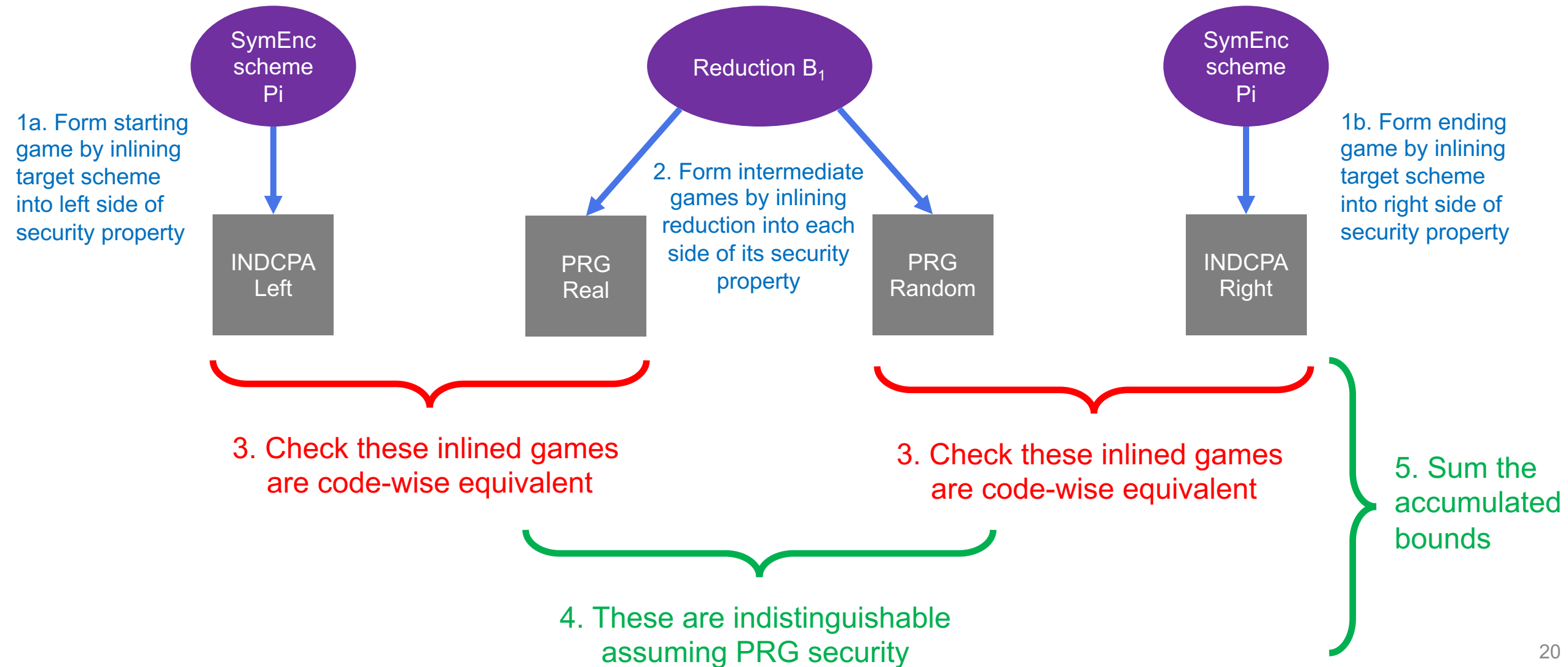
- Initialize routine is run once
- Adversary can call any oracles, many times, in any order

A symmetric encryption scheme  $\Sigma$  is CPA-secure if the following two games are indistinguishable:

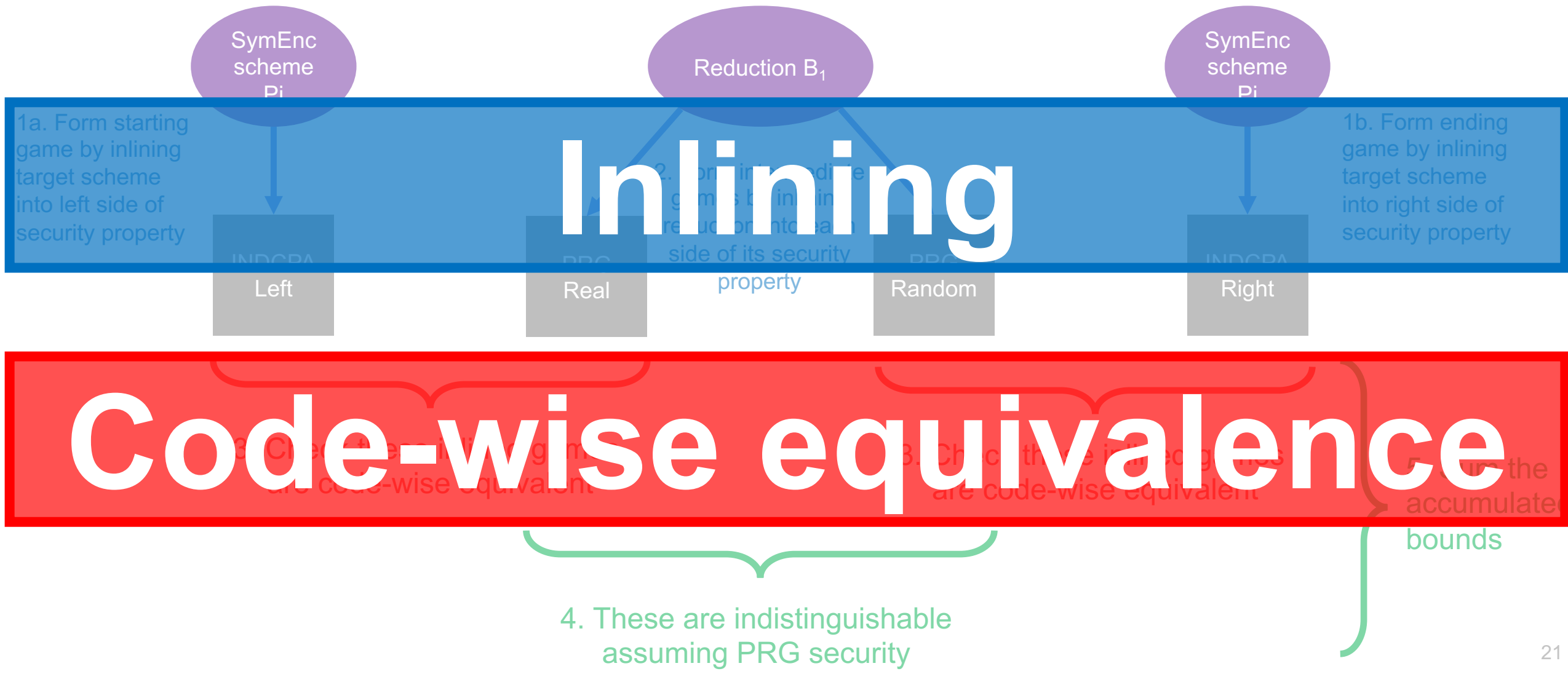
$\mathcal{L}_{\text{cpa-L}}^\Sigma$
$k \leftarrow \Sigma.\text{KeyGen}$
$\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M}):$
$c := \Sigma.\text{Enc}(k, m_L)$
return $c$

$\mathcal{L}_{\text{cpa-R}}^\Sigma$
$k \leftarrow \Sigma.\text{KeyGen}$
$\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M}):$
$c := \Sigma.\text{Enc}(k, m_R)$
return $c$

# Mechanics of checking a game-hopping proof



# Mechanics of checking a game-hopping proof



# Code-wise equivalence

## via automated program canonicalization

Check if two games are equivalent by transforming their source code into a “**canonical form**”, then checking if their canonical forms are equal as strings

Transformations must be semantics-preserving

- e.g., rename variables, sort statements, eliminate unused lines

# ProofFrog Limitations

- Automation fails => no manual recourse
- Tool is immature: new proofs often require engine extensions
- Limited expressivity
  - Proof techniques
  - Mathematical constructions
- Limited mathematical/probabilistic reasoning
- No verification of ProofFrog engine
- Not yet peer-reviewed
- Generative AI used for engine development

# Use in classes

Tempting to use in class to force students to learn to strictly follow game-hop proof mechanics

Currently being tried (not for grades) at:

- TUE by Andreas Hülsing & Kathrin Hövelmanns
- FAU Erlangen-Nürnberg by Paul Rösler

## Resources available:

- `pip install proof_frog`
- ProofFrog manual with tutorial and worked examples
- Library of basic primitives, security definitions, proofs
- Private examples repository for instructors
- I am happy to help!

# Generative AI for proof development

Have an MCP server for coding agents to interact with ProofFrog engine

MCP: Model Context Protocol

- Way for chat agents to interact with services

```
> Write a ProofFrog proof
> that the symmetric
> encryption scheme
>  $ctxt = (nonce,$ 
>  $PRF(k, nonce) \text{ xor } msg)$ 
> is IND-CPA secure.
```

</> Untitled-1

Write a ProofFrog proof that the symmetric encryption scheme  $ctxt = (nonce, PRF(k, nonce) \oplus msg)$  is IND-CPA secure.

- Thinking
- I'll build this. The construction  $ctxt = (nonce, PRF(k, nonce) \oplus msg)$  is the standard "randomized counter-mode-style" PRF encryption. Plan: (1) define a `NoncePRFEnc(PR F)` scheme implementing the `SymEnc` primitive, (2) prove `INDCPA$MultiChal` (real-or-random; trivially implies IND-CPA) using two assumption hops — `PRFSecurity` to swap PRF→random function, then `UniqueSampling` so the RF is queried on unique nonces and its output XORed with msg becomes uniform. Let me check a couple more conventions before writing.

• Write NoncePRFEnc.scheme

28 lines

```
// Nonce-based symmetric encryption from a PRF:
// Enc(k, m): sample nonce n; return [n, F.evaluate(k, n) XOR m].
// Dec(k, c): return c[1] XOR F.evaluate(k, c[0]).
```

Scheme NoncePRFEnc(PR F) extends SymEnc {

```
Set Message = BitString<F.out>;
Set Ciphertext = [BitString<F.in>, BitString<F.out>];
Set Key = BitString<F.lambda>;
```

```
Key KeyGen() {
  Key k <- Key;
  return k;
}
```

```
Ciphertext Enc(Key k, Message m) {
  BitString<F.in> nonce <- BitString<F.in>;
  return [nonce, F.evaluate(k, nonce) + m];
}
```

```
deterministic Message? Dec(Key k, Ciphertext c) {
  return c[1] + F.evaluate(k, c[0]);
}
```

• Write NoncePRFEnc\_INDCPA\$MultiChal.proof

75 lines

```
// NoncePRFEnc satisfies multi-challenge IND-CPA$ (real-or-random) security.
// Assumes: PRFSecurity(F) and UniqueSampling(BitString<F.in>).
//
```

[initial proof failed, it read some more examples & engine source code]

• Edit NoncePRFEnc\_INDCPA\$MultiChal.proof

Added 3 lines

```
UniqueSampling(BitString<F.in>).Replacement compose R_Uniq(F) against INDCPA$MultiChal
(E).Adversary;
UniqueSampling(BitString<F.in>).NoReplacement compose R_Uniq(F) against INDCPA$MultiChal
(E).Adversary;
UniqueSampling(BitString<F.in>).Replacement compose R_Uniq(F) against INDCPA$MultiChal
(E).Adversary;
UniqueSampling(BitString<F.in>).NoReplacement compose R_Uniq(F) against INDCPA$MultiChal
(E).Adversary;
```

• Bash Run updated proof

```
IN python -m proof_frog prove 'examples/Proofs/SymEnc/NoncePRFEnc_INDCPA$MultiChal.proof' 2>&1 | tail
OUT      -> INDCPA$MultiChal(E).Random

Proof Succeeded!
```

• The proof verifies. Files produced:

- [examples/Schemes/SymEnc/NoncePRFEnc.scheme](#) — the construction  $Enc(k, m) = (n, F(k, n) \oplus m)$  with random nonce, deterministic decryption.
- [examples/Proofs/SymEnc/NoncePRFEnc\\_INDCPA\\$MultiChal.proof](#) — proves multi-challenge IND-CPA\$ (which trivially implies IND-CPA) assuming `PRFSecurity(F)` and `UniqueSampling(BitString<F.in>)`.

**Proof outline (7 hops):** swap PRF → random function via `R_PRF` and the PRF assumption; swap uniform nonce → unique nonce via `R_Uniq` and `UniqueSampling`, which lets the engine apply `FreshInputRFToUniform` to replace `RF(nonce)` with a fresh uniform string; then a second `R_Final` reduction with the same RF-field shape but no RF call lets us run `UniqueSampling` in reverse to drop the bookkeeping, leaving a uniform random ciphertext that matches `INDCPA$.Random`. The double `UniqueSampling` cycle is needed because the engine keeps the RF field alive while its `.domain` is referenced as exclusion-set bookkeeping; without the second cycle, hop 5 fails to canonicalize to the bookkeeping-free uniform-tuple form.

Note: the `UniqueSampling` assumption carries a birthday-bound statistical loss of  $\sim q^2/2^{|F.in|}$ , so the result is meaningful only when the nonce space is large enough that nonces don't collide with high probability — standard for a randomized nonce-based scheme.

Total runtime: 11 minutes

# ProofFrog <sup>to</sup> ~~vs.~~ EasyCrypt

EasyCrypt is much more expressive and has a much higher degree of assurance

ProofFrog will never catch up, but maybe can fill a different niche, benefitting from and complementing EasyCrypt

- Can we **export** ProofFrog **schemes & games** to EasyCrypt?
- Can we **export** ProofFrog **proofs** to EasyCrypt?
- Can we **complete gaps** in ProofFrog proofs in EasyCrypt?

# CFRG hybrid KEMs analysis using ProofFrog

Crypto Forum  
Internet-Draft  
Intended status: Informational  
Expires: 3 September 2026

D. Connolly  
SandboxAQ  
R. Barnes  
Cisco  
P. Grubbs  
University of Michigan  
2 March 2026

Hybrid PQ/T Key Encapsulation Mechanisms  
draft-irtf-cfrg-hybrid-kems-10

# CFRG hybrid KEM constructions

	Traditional is a group	Traditional is a KEM
<b>Universal combiner</b>	<p><b>UG</b></p> $ss = H(ss_{PQ}, ss_T, ct_{PQ}, ct_T, pk_{PQ}, pk_T)$ <p>like StarFortress [CG26]</p>	<p><b>UK</b></p> $ss = H(ss_{PQ}, ss_T, ct_{PQ}, ct_T, pk_{PQ}, pk_T)$ <p>like [GHP18]</p>
<p><b>C2PRI combiner</b></p> <ul style="list-style-type: none"> <li>omit <math>ct_{PQ}, pk_{PQ}</math></li> <li>assume PQ KEM is ciphertext 2<sup>nd</sup> preimage resistant</li> </ul>	<p><b>CG</b></p> $ss = H(ss_{PQ}, ss_T, ct_T, pk_T)$ <p>like X-Wing [BCD+24] / QSF StarFighters [CHH+25]</p>	<p><b>CK</b></p> $ss = H(ss_{PQ}, ss_T, ct_T, pk_T)$ <p>like StarHunters [COSS26]</p>

<https://datatracker.ietf.org/doc/draft-irtf-cfrg-hybrid-kems/>

[BCD+24] Barbosa, Connolly, Duarte, Kaiser, Shwabe, Varner, Westerbaan. IACR CIC 2024.

[CG26] Connolly, Grubbs. Eprint 2026/125.

[CHH+25] Connolly, Hövelmanns, Hülsing, Kousidis, Meijers. Eprint 2025/1397.

[COSS26] Connolly, Ounsworth, Schmieg, Stebila. Eprint 2026/427.

[GHP18] Giacon, Heuer, et al. PKC 2018

**x 2**

**Seeded:** sk is a single seed

**Expanded:** full sk's

# Intended security properties

## Correctness

## Hybrid IND-CCA security

- IND-CCA if PQ component is IND-CCA
- IND-CCA if traditional component is IND-CCA

## Binding properties [CDM24]

Session key  $K$  binds PK or CT

- “Changing PK or CT changes  $K$ ”
- MAL: adversary-supplied keypair
- LEAK: honest keypair, given to adversary
- HON: honest keypair, decaps oracle

Targets:

- LEAK-BIND-K-PK
- LEAK-BIND-K-CT

# Preliminary results

	CG seeded	CG expanded	CK seeded	CK expanded	UG seeded	UG expanded	UK seeded	UK expanded
Traditional	group	group	KEM	KEM	group	group	KEM	KEM
Combiner	C2PRI	C2PRI	C2PRI	C2PRI	universal	universal	universal	universal
Correctness	✓	✓	✓	✓	✓	✓	✓	✓
IND-CCA traditional	✓ ROM	✓ ROM	✓	✓	✓ ROM	✓ ROM	✓	✓
IND-CCA PQ	✓	✓	✓	✓	✓	✓	✓	✓
LEAK-BIND -K-CT	⚠️ std model ✓ PRG ROM	✓	⚠️ std model ✓ PRG ROM	✓	✓	✓	✓	✓
LEAK-BIND -K-PK	⚠️ std model ✓ PRG ROM	✓	⚠️ std model ✓ PRG ROM	✓	✓	✓	✓	✓
HON-BIND -K-CT	✓ std model	implied	✓ std model	implied	implied	implied	implied	implied
HON-BIND -K-PK	✓ std model	implied	✓ std model	implied	implied	implied	implied	implied

# {CG, CK}-seeded binding in standard model

LEAK-BIND  
reveals  $sk$  to  
the adversary

## Problem for CG & CK seeded:

- $pk_{PQ}$  and  $ct_{PQ}$  are not included in hash
- So binding of hybrid has to come via reduction to binding of underlying PQ KEM
- Reduction receives PQ KEM's secret key (either as seed or expanded)
- But reduction has to **simulate** a hybrid secret key which is a single **master seed that expands to the target PQ KEM secret key**

Can show **LEAK-BIND**  
in **ROM**:

- Program the PRG

Can show **HON-BIND**  
in **std model**:

- Adversary only gets decaps oracle, so standard-model reduction can simulate via passthrough

AFAIK  
these subtleties  
were not known  
before this work.

# Proof creation process

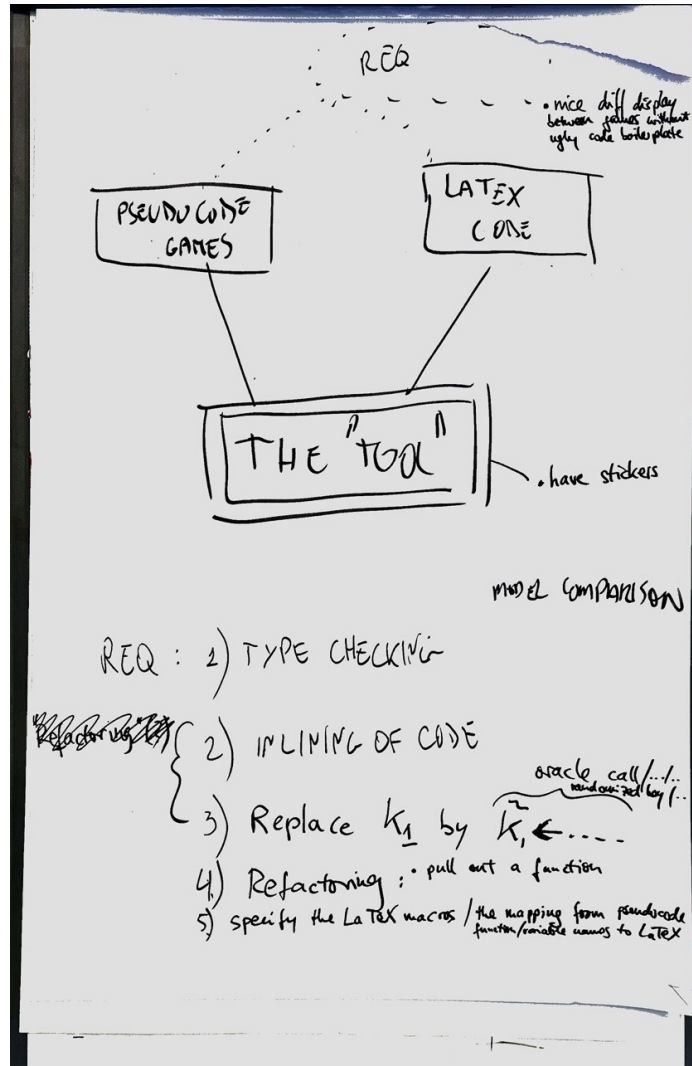
1. Have Claude read the Internet-Draft to create the schemes
2. Carefully review and edit scheme definitions and security properties
  - \*-BIND-\* properties as left/right games
3. Help Claude make the proofs
  - ProofFrog example corpus includes hybrid IND-CCA security of [GHP18] & StarFortress
  - Multiple engine extensions required
  - Approximately 20 hours of working with Claude for engine extension + proofs
  - Figure out shape of results for seeded binding and prompt Claude
  - Claude mostly figured out the sequence of games and the reductions
4. Carefully review theorem statements

# Wrapping up

# Hierarchy of tool capabilities

1. LaTeX management
2. Basic inlining & typechecking
3. More advanced typechecking
4. Exportable to formal verification

# Requirements satisfied?



# Reflection & conclusion

## Problem context

Hierarchy of proof tools for pen-and-paper cryptographers

## Discussion

What is useful to us as proof authors?

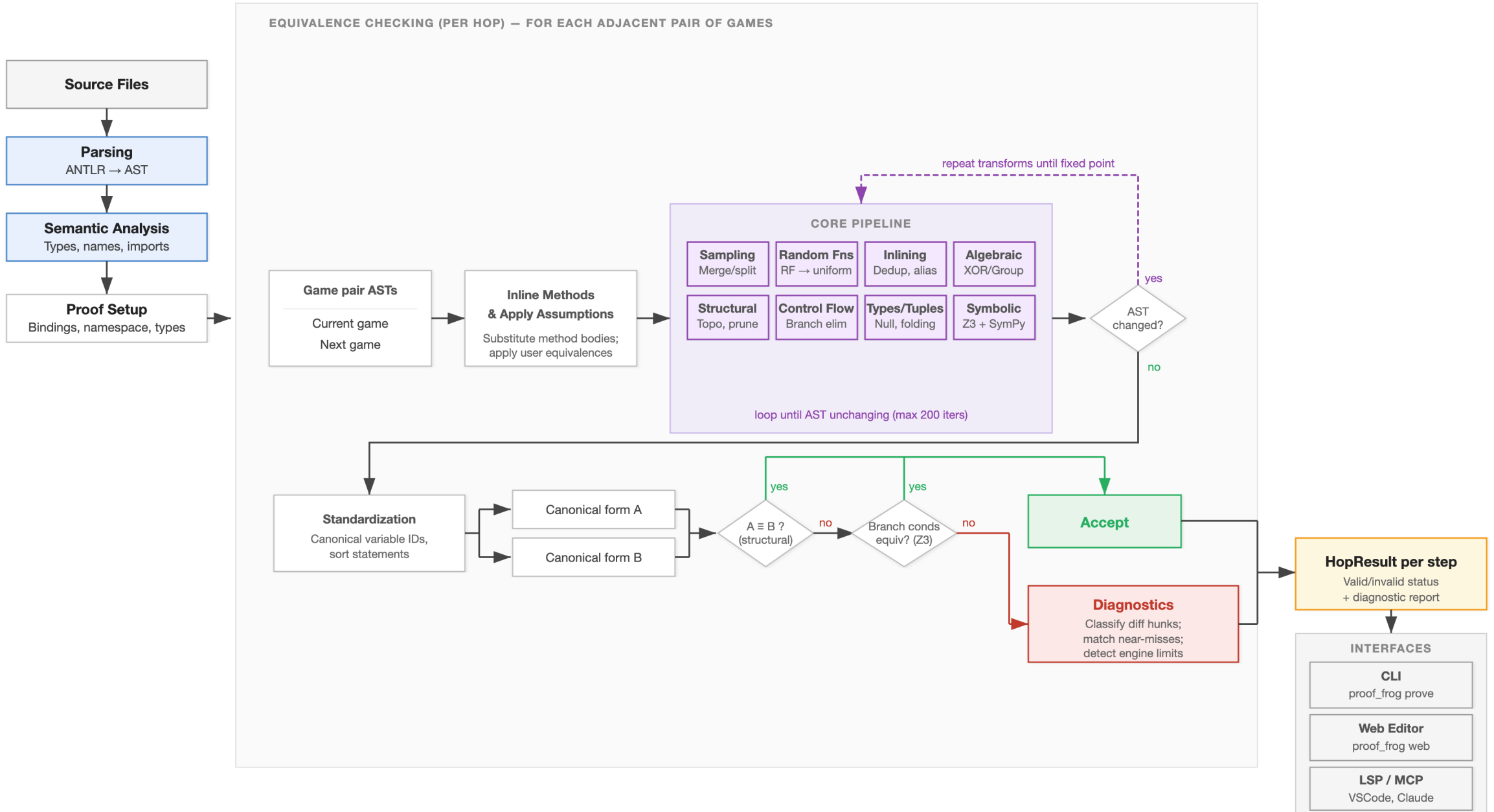
What would we like as proof readers?

## Some tools

- TeXFrog for LaTeX management?
- ProofFrog as an educational tool?
- ProofFrog as a research tool?
  - “moderate assurance”?
  - export to high assurance?

# Appendix

# ProofFrog engine



# ProofFrog sample: KEM.primitive

```
1  Primitive KEM(Int Nseed, Int Nss, Int Nct, Int Nek, Set SharedSecretSpace, Set CiphertextSpace,  
  Set EncapsKeySpace, Set DecapsKeySpace) {  
2    Int Nseed = Nseed;  
3    Int Nss = Nss;  
4    Int Nct = Nct;  
5    Int Nek = Nek;  
6    Set SharedSecret = SharedSecretSpace;  
7    Set Ciphertext = CiphertextSpace;  
8    Set EncapsKey = EncapsKeySpace;  
9    Set DecapsKey = DecapsKeySpace;  
10  
11   [EncapsKey, DecapsKey] KeyGen();  
12   deterministic [EncapsKey, DecapsKey] DeriveKeyPair(BitString<Nseed> seed);  
13   [SharedSecret, Ciphertext] Encaps(EncapsKey ek);  
14   deterministic SharedSecret Decaps(DecapsKey dk, Ciphertext ct);  
15   deterministic injective BitString<Nss> EncodeSharedSecret(SharedSecret ss);  
16   deterministic injective BitString<Nct> EncodeCiphertext(Ciphertext ct);  
17   deterministic injective BitString<Nek> EncodeEncapsKey(EncapsKey ek);  
18 }
```

# {LEAK,HON}-BIND-K-{CT,PK}

```
X-BIND-P-QAKEM :  
-----  
sk0, pk0 ← KeyGen()  
sk1, pk1 ← KeyGen()  
if pk ∈ Q : b ← 1  
else if pk ∈ P : b ← 0  
else : b ∈ {0, 1}, st ← A()  
sk1, pk1 ← skb, pkb  
if X = HON : ct0, ct1 ← ADb'(skb, ·)(pk0, pk1, st)  
if X = LEAK : ct0, ct1 ← A(pk0, sk0, pk1, sk1, st)  
k0 ← KEM.Decaps(sk0, pk0, ct0)  
k1 ← KEM.Decaps(sk1, pk1, ct1)  
if k0 = ⊥ ∨ k1 = ⊥ : return 0  
// A wins if ¬((∀x ∈ P . x0 = x1) ⇒ (∀y ∈ Q . y0 = y1))  
return ∀x ∈ P . x0 = x1 ∧ ∃y ∈ Q . y0 ≠ y1
```

Figure 5: Generic game for our new binding notions  $X$ -BIND- $P$ - $Q$  for  $X \in \{HON, LEAK\}$ .

# LEAK-BIND-K-CT

Game LEAK-BIND-K-CT.Breakable(K)

Game LEAK-BIND-K-CT.Unbreakable(K)

Initialize()

---

1 :  $(pk_0, sk_0) \leftarrow K.\text{KeyGen}()$   
2 :  $(pk_1, sk_1) \leftarrow K.\text{KeyGen}()$   
3 : **return**  $(pk_0, sk_0, pk_1, sk_1)$

Challenge( $ct_0, ct_1$ )

---

1 :  $k_0 \leftarrow K.\text{Decaps}(sk_0, ct_0)$   
2 :  $k_1 \leftarrow K.\text{Decaps}(sk_1, ct_1)$   
3 : **return**  $k_0 = k_1 \wedge ct_0 \neq ct_1$

Initialize()

---

1 :  $(pk_0, sk_0) \leftarrow K.\text{KeyGen}()$   
2 :  $(pk_1, sk_1) \leftarrow K.\text{KeyGen}()$   
3 : **return**  $(pk_0, sk_0, pk_1, sk_1)$

Challenge( $ct_0, ct_1$ )

---

1 :  $k_0 \leftarrow K.\text{Decaps}(sk_0, ct_0)$   
2 :  $k_1 \leftarrow K.\text{Decaps}(sk_1, ct_1)$   
3 : **return false**

# ProofFrog sample: LEAK-BIND-K-CT . game

```
Game Breakable(KEM K) {
    K.DecapsKey dk0;
    K.DecapsKey dk1;

    [K.EncapsKey, K.DecapsKey, K.EncapsKey, K.DecapsKey] Initialize() {
        [K.EncapsKey, K.DecapsKey] kp0 = K.KeyGen();
        [K.EncapsKey, K.DecapsKey] kp1 = K.KeyGen();
        dk0 = kp0[1];
        dk1 = kp1[1];
        return [kp0[0], kp0[1], kp1[0], kp1[1]];
    }

    Bool Challenge(K.Ciphertext ct0, K.Ciphertext ct1) {
        K.SharedSecret k0 = K.Decaps(dk0, ct0);
        K.SharedSecret k1 = K.Decaps(dk1, ct1);
        return k0 == k1 && ct0 != ct1;
    }
}
```

```
Game Unbreakable(KEM K) {
    K.DecapsKey dk0;
    K.DecapsKey dk1;

    [K.EncapsKey, K.DecapsKey, K.EncapsKey, K.DecapsKey] Initialize() {
        [K.EncapsKey, K.DecapsKey] kp0 = K.KeyGen();
        [K.EncapsKey, K.DecapsKey] kp1 = K.KeyGen();
        dk0 = kp0[1];
        dk1 = kp1[1];
        return [kp0[0], kp0[1], kp1[0], kp1[1]];
    }

    Bool Challenge(K.Ciphertext ct0, K.Ciphertext ct1) {
        K.SharedSecret k0 = K.Decaps(dk0, ct0);
        K.SharedSecret k1 = K.Decaps(dk1, ct1);
        return false;
    }
}
```

# LEAK-BIND-K-PK

Game LEAK-BIND-K-PK.Breakable(K)

Game LEAK-BIND-K-PK.Unbreakable(K)

Initialize()

---

- 1 :  $(pk_0, sk_0) \leftarrow K.\text{KeyGen}()$
- 2 :  $(pk_1, sk_1) \leftarrow K.\text{KeyGen}()$
- 3 : **return**  $(pk_0, sk_0, pk_1, sk_1)$

Challenge( $ct_0, ct_1$ )

---

- 1 :  $k_0 \leftarrow K.\text{Decaps}(sk_0, ct_0)$
- 2 :  $k_1 \leftarrow K.\text{Decaps}(sk_1, ct_1)$
- 3 : **return**  $k_0 = k_1 \wedge pk_0 \neq pk_1$

Initialize()

---

- 1 :  $(pk_0, sk_0) \leftarrow K.\text{KeyGen}()$
- 2 :  $(pk_1, sk_1) \leftarrow K.\text{KeyGen}()$
- 3 : **return**  $(pk_0, sk_0, pk_1, sk_1)$

Challenge( $ct_0, ct_1$ )

---

- 1 :  $k_0 \leftarrow K.\text{Decaps}(sk_0, ct_0)$
- 2 :  $k_1 \leftarrow K.\text{Decaps}(sk_1, ct_1)$
- 3 : **return false**

# ProofFrog sample: UK\_seedbased\_scheme

```
deterministic [EncapsKey, DecapsKey] DeriveKeyPair(BitString<Nseed> seed) {
    BitString<G.lambda + G.stretch> seed_full = G.evaluate(seed);
    BitString<KEM_PQ.Nseed> seed_PQ = seed_full[0 : KEM_PQ.Nseed];
    BitString<KEM_T.Nseed> seed_T = seed_full[KEM_PQ.Nseed : KEM_PQ.Nseed + KEM_T.Nseed];
    [KEM_PQ.EncapsKey, KEM_PQ.DecapsKey] pq_keypair = KEM_PQ.DeriveKeyPair(seed_PQ);
    [KEM_T.EncapsKey, KEM_T.DecapsKey] t_keypair = KEM_T.DeriveKeyPair(seed_T);
    return [[pq_keypair[0], t_keypair[0]], seed];
}

[SharedSecret, Ciphertext] Encaps(EncapsKey ek) {
    KEM_PQ.EncapsKey ek_PQ = ek[0];
    KEM_T.EncapsKey ek_T = ek[1];
    [KEM_PQ.SharedSecret, KEM_PQ.Ciphertext] pq_enc = KEM_PQ.Encaps(ek_PQ);
    [KEM_T.SharedSecret, KEM_T.Ciphertext] t_enc = KEM_T.Encaps(ek_T);
    KEM_PQ.SharedSecret ss_PQ = pq_enc[0];
    KEM_PQ.Ciphertext ct_PQ = pq_enc[1];
    KEM_T.SharedSecret ss_T = t_enc[0];
    KEM_T.Ciphertext ct_T = t_enc[1];
    BitString<H.Nin> kdf_in = KEM_PQ.EncodeSharedSecret(ss_PQ) || KEM_T.EncodeSharedSecret
(ss_T) || KEM_PQ.EncodeCiphertext(ct_PQ) || KEM_T.EncodeCiphertext(ct_T) || KEM_PQ.
EncodeEncapsKey(ek_PQ) || KEM_T.EncodeEncapsKey(ek_T) || L.get();
    SharedSecret ss = H.evaluate(kdf_in);
    return [ss, [ct_PQ, ct_T]];
}
```



# ProofFrog sample: UK\_seedbased\_LEAK\_BIND\_K\_CT.proof

```
Reduction R(KEM KEM_PQ, KEM KEM_T, PRG G, KDF H, Label L, UK_seedbased hybrid)
compose KDFCollisionResistance(H) against LEAK_BIND_K_CT(hybrid).Adversary {

  BitString<G.lambda> seed0;
  BitString<G.lambda> seed1;

  [hybrid.EncapsKey, hybrid.DecapsKey, hybrid.EncapsKey, hybrid.DecapsKey] Initialize() {
    [hybrid.EncapsKey, hybrid.DecapsKey] kp0 = hybrid.KeyGen();
    [hybrid.EncapsKey, hybrid.DecapsKey] kp1 = hybrid.KeyGen();
    seed0 = kp0[1];
    seed1 = kp1[1];
    return [kp0[0], kp0[1], kp1[0], kp1[1]];
  }

  Bool Challenge(hybrid.Ciphertext ct0, hybrid.Ciphertext ct1) {
    BitString<G.lambda + G.stretch> seed_full_0 = G.evaluate(seed0);
    BitString<KEM_PQ.Nseed> seed_PQ_0 = seed_full_0[0 : KEM_PQ.Nseed];
    BitString<KEM_T.Nseed> seed_T_0 = seed_full_0[KEM_PQ.Nseed : KEM_PQ.Nseed + KEM_T.Nseed];
    [KEM_PQ.EncapsKey, KEM_PQ.DecapsKey] pq_kp_0 = KEM_PQ.DeriveKeyPair(seed_PQ_0);
    [KEM_T.EncapsKey, KEM_T.DecapsKey] t_kp_0 = KEM_T.DeriveKeyPair(seed_T_0);
    KEM_PQ.Ciphertext ct_PQ_0 = ct0[0];
    KEM_T.Ciphertext ct_T_0 = ct0[1];
    KEM_PQ.SharedSecret ss_PQ_0 = KEM_PQ.Decaps(pq_kp_0[1], ct_PQ_0);
    KEM_T.SharedSecret ss_T_0 = KEM_T.Decaps(t_kp_0[1], ct_T_0);
    BitString<H.Nin> kdf_in_0 = KEM_PQ.EncodeSharedSecret(ss_PQ_0) || KEM_T.EncodeSharedSecret(ss_T_0) || KEM_PQ.
    EncodeCiphertext(ct_PQ_0) || KEM_T.EncodeCiphertext(ct_T_0) || KEM_PQ.EncodeEncapsKey(pq_kp_0[0]) || KEM_T.
    EncodeEncapsKey(t_kp_0[0]) || L.get();

    BitString<G.lambda + G.stretch> seed_full_1 = G.evaluate(seed1);
    BitString<KEM_PQ.Nseed> seed_PQ_1 = seed_full_1[0 : KEM_PQ.Nseed];
    BitString<KEM_T.Nseed> seed_T_1 = seed_full_1[KEM_PQ.Nseed : KEM_PQ.Nseed + KEM_T.Nseed];
    [KEM_PQ.EncapsKey, KEM_PQ.DecapsKey] pq_kp_1 = KEM_PQ.DeriveKeyPair(seed_PQ_1);
    [KEM_T.EncapsKey, KEM_T.DecapsKey] t_kp_1 = KEM_T.DeriveKeyPair(seed_T_1);
    KEM_PQ.Ciphertext ct_PQ_1 = ct1[0];
    KEM_T.Ciphertext ct_T_1 = ct1[1];
    KEM_PQ.SharedSecret ss_PQ_1 = KEM_PQ.Decaps(pq_kp_1[1], ct_PQ_1);
    KEM_T.SharedSecret ss_T_1 = KEM_T.Decaps(t_kp_1[1], ct_T_1);
    BitString<H.Nin> kdf_in_1 = KEM_PQ.EncodeSharedSecret(ss_PQ_1) || KEM_T.EncodeSharedSecret(ss_T_1) || KEM_PQ.
    EncodeCiphertext(ct_PQ_1) || KEM_T.EncodeCiphertext(ct_T_1) || KEM_PQ.EncodeEncapsKey(pq_kp_1[0]) || KEM_T.
    EncodeEncapsKey(t_kp_1[0]) || L.get();

    if (ct0 == ct1) {
      return false;
    }
    return challenger.Challenge(kdf_in_0, kdf_in_1);
  }
}
```

# ProofFrog sample execution:

## UK\_seedbased\_LEAK\_BIND\_K\_CT.proof

[proof\\_frog prove UK\\_seedbased\\_LEAK\\_BIND\\_K\\_CT.proof](#)  
Type checking...

Theorem: LEAK\_BIND\_K\_CT(hybrid)

```
Step 1/3 LEAK_BIND_K_CT(hybrid).Breakable -> KDFCollisionResistance(H).Breakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid) ... ok
Step 2/3 KDFCollisionResistance(H).Breakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid)
        -> KDFCollisionResistance(H).Unbreakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid) ... by assumption
Step 3/3 KDFCollisionResistance(H).Unbreakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid) -> LEAK_BIND_K_CT(hybrid).Unbreakable ... ok
```

Step	Hop	Type	Result
1	LEAK_BIND_K_CT(hybrid).Breakable -> KDFCollisionResistance(H).Breakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid)	equivalence	ok
2	KDFCollisionResistance(H).Breakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid) -> KDFCollisionResistance(H).Unbreakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid)	assumption	assume
3	KDFCollisionResistance(H).Unbreakable compose R(KEM_PQ, KEM_T, G, H, L, hybrid) -> LEAK_BIND_K_CT(hybrid).Unbreakable	equivalence	ok

Proof Succeeded!

# ProofFrog sample execution:

## UK\_seedbased\_LEAK\_BIND\_K\_CT.proof

===STEP 2===

Current: KDFCollisionResistance(H).Breakable compose R(KEM\_PQ, KEM\_T, G, H, L, hybrid)  
Hop To: KDFCollisionResistance(H).Unbreakable compose R(KEM\_PQ, KEM\_T, G, H, L, hybrid)

Valid by assumption

Step 2/3 KDFCollisionResistance(H).Breakable compose R(KEM\_PQ, KEM\_T, G, H, L, hybrid)  
-> KDFCollisionResistance(H).Unbreakable compose R(KEM\_PQ, KEM\_T, G, H, L, hybrid) ... by assumption

CURRENT

Game Inlined() {

```
[[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] Initialize() {
  BitString<kem_pq_nseed> v1 <- BitString<kem_pq_nseed>;
  BitString<kem_pq_nseed + kem_t_nseed> v2 = G.evaluate(v1);
  BitString<kem_pq_nseed> v3 <- BitString<kem_pq_nseed>;
  BitString<kem_pq_nseed + kem_t_nseed> v4 = G.evaluate(v3);
  return [[KEM_PQ.DeriveKeyPair(v2[0 : kem_pq_nseed])][0], KEM_T.DeriveKeyPair(v2[kem_pq_nseed : kem_pq_nseed + kem_t_nseed])[0]], v1, [KEM_PQ.DeriveKeyPair(v4[0 : kem_pq_nseed])][0], KEM_T.DeriveKeyPair(v4[kem_pq_nseed : kem_pq_nseed + kem_t_nseed])[0]], v3];
}
Bool Challenge([KEMPQCiphertextSpace, KEMTCiphertextSpace] ct0, [KEMPQCiphertextSpace, KEMTCiphertextSpace] ct1) {
  return false;
}
}
```

NEXT

Game Unbreakable() {

```
[[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] Initialize() {
  BitString<kem_pq_nseed> v1 <- BitString<kem_pq_nseed>;
  BitString<kem_pq_nseed + kem_t_nseed> v2 = G.evaluate(v1);
  BitString<kem_pq_nseed> v3 <- BitString<kem_pq_nseed>;
  BitString<kem_pq_nseed + kem_t_nseed> v4 = G.evaluate(v3);
  return [[KEM_PQ.DeriveKeyPair(v2[0 : kem_pq_nseed])][0], KEM_T.DeriveKeyPair(v2[kem_pq_nseed : kem_pq_nseed + kem_t_nseed])[0]], v1, [KEM_PQ.DeriveKeyPair(v4[0 : kem_pq_nseed])][0], KEM_T.DeriveKeyPair(v4[kem_pq_nseed : kem_pq_nseed + kem_t_nseed])[0]], v3];
}
Bool Challenge([KEMPQCiphertextSpace, KEMTCiphertextSpace] ct0, [KEMPQCiphertextSpace, KEMTCiphertextSpace] ct1) {
  return false;
}
}
```

Inline Success!

-----

EXPLORER

- easycrypt
- games
- primitives
- ▼ proofs
  - CG
  - CK
  - Generic
  - UG
  - ▼ UK
    - UK\_expanded\_Correctn...
    - UK\_expanded\_INDCCA...
    - UK\_expanded\_INDCCA...
    - UK\_expanded\_LEAK\_BI...
    - UK\_expanded\_LEAK\_BI...
    - UK\_seedbased\_Correct...
    - UK\_seedbased\_INDCC...
    - UK\_seedbased\_INDCC...
    - UK\_seedbased\_LEAK\_...
    - UK\_seedbased\_LEAK\_...
- schemes
- README.md

UK\_seedbased... x [inline] KDFColli... x

```
Previous Game — Inlined
1 Game Inlined() {
2   BitString<kem_pq_nseed> seed0;
3   BitString<kem_pq_nseed> seed1;
4   [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace,
5     BitString<kem_pq_nseed> hybrid.KeyGen@seed0 <- BitString<kem_pq_nseed>;
6     BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full1 = G.evaluate(hybr:
7     BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ1 = hybrid.DeriveKeyPair@seed_full1
8     BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T1 = hybrid.DeriveKeyPair@seed_full1[k
9     [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair1 = KEM_PQ.Dei
10    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair1 = KEM_T.Derive
11    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp0 = [[hybrid.Der:
12    BitString<kem_pq_nseed> hybrid.KeyGen@seed7 <- BitString<kem_pq_nseed>;
13    BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full8 = G.evaluate(hybr:
14    BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ8 = hybrid.DeriveKeyPair@seed_full8
15    BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T8 = hybrid.DeriveKeyPair@seed_full8[k
16    [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair8 = KEM_PQ.Dei
17    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair8 = KEM_T.Derive
18    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp1 = [[hybrid.Der:
```

```
Current Game — Inlined
1 Game Inlined() {
2   BitString<kem_pq_nseed> seed0;
3   BitString<kem_pq_nseed> seed1;
4   [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace,
5     BitString<kem_pq_nseed> hybrid.KeyGen@seed0 <- BitString<kem_pq_nseed>;
6     BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full1 = G.evaluate(hybr:
7     BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ1 = hybrid.DeriveKeyPair@seed_full1
8     BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T1 = hybrid.DeriveKeyPair@seed_full1[k
9     [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair1 = KEM_PQ.Dei
10    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair1 = KEM_T.Derive
11    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp0 = [[hybrid.Der:
12    BitString<kem_pq_nseed> hybrid.KeyGen@seed7 <- BitString<kem_pq_nseed>;
13    BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full8 = G.evaluate(hybr:
14    BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ8 = hybrid.DeriveKeyPair@seed_full8
15    BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T8 = hybrid.DeriveKeyPair@seed_full8[k
16    [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair8 = KEM_PQ.Dei
17    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair8 = KEM_T.Derive
18    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp1 = [[hybrid.Der:
```

Reduction

```
1 Reduction R() compose KDFCollisionResistance(H) against LEAK_BIND_K_CT(hybrid).Adversary {
2   BitString<kem_pq_nseed> seed0;
3   BitString<kem_pq_nseed> seed1;
4   [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace,
5     BitString<kem_pq_nseed> hybrid.KeyGen@seed0 <- BitString<kem_pq_nseed>;
6     BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full1 = G.evaluate(hybr:
7     BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ1 = hybrid.DeriveKeyPair@seed_full1
8     BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T1 = hybrid.DeriveKeyPair@seed_full1[k
9     [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair1 = KEM_PQ.Dei
10    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair1 = KEM_T.Derive
11    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp0 = [[hybrid.Der:
12    BitString<kem_pq_nseed> hybrid.KeyGen@seed7 <- BitString<kem_pq_nseed>;
13    BitString<kem_pq_nseed + kem_t_nseed> hybrid.DeriveKeyPair@seed_full8 = G.evaluate(hybr:
14    BitString<kem_pq_nseed> hybrid.DeriveKeyPair@seed_PQ8 = hybrid.DeriveKeyPair@seed_full8
15    BitString<kem_t_nseed> hybrid.DeriveKeyPair@seed_T8 = hybrid.DeriveKeyPair@seed_full8[k
16    [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] hybrid.DeriveKeyPair@pq_keypair8 = KEM_PQ.Dei
17    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] hybrid.DeriveKeyPair@t_keypair8 = KEM_T.Derive
18    [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>] kp1 = [[hybrid.Der:
```

Previous Game — Canonical Form

```
1 Game Inlined() {
2   BitString<kem_pq_nek> field1;
3   BitString<kem_pq_nek> field2;
4   BitString<kem_t_nek> field3;
5   BitString<kem_t_nek> field4;
6   KEMPQDecapsKeySpace field5;
7   KEMPQDecapsKeySpace field6;
8   KEMTDecapsKeySpace field7;
9   KEMTDecapsKeySpace field8;
10  [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace,
11    BitString<kem_pq_nseed> v1 <- BitString<kem_pq_nseed>;
12    BitString<kem_pq_nseed + kem_t_nseed> v2 = G.evaluate(v1);
13    [KEMPQEncapsKeySpace, KEMPQDecapsKeySpace] v3 = KEM_PQ.DeriveKeyPair(v2[0 : kem_pq_nseed
14    KEMPQEncapsKeySpace v4 = v3[0];
15    [KEMTEncapsKeySpace, KEMTDecapsKeySpace] v5 = KEM_T.DeriveKeyPair(v2[kem_pq_nseed : kem_
16    KEMTEncapsKeySpace v6 = v5[0];
17    BitString<kem_pq_nseed> v7 <- BitString<kem_pq_nseed>;
18    BitString<kem_pq_nseed + kem_t_nseed> v8 = G.evaluate(v7);
```

Current Game — Canonical Form

```
1 Game Inlined() {
2   [[KEMPQEncapsKeySpace, KEMTEncapsKeySpace], BitString<kem_pq_nseed>, [KEMPQEncapsKeySpace,
3     BitString<kem_pq_nseed> v1 <- BitString<kem_pq_nseed>;
4     BitString<kem_pq_nseed + kem_t_nseed> v2 = G.evaluate(v1);
5     BitString<kem_pq_nseed> v3 <- BitString<kem_pq_nseed>;
6     BitString<kem_pq_nseed + kem_t_nseed> v4 = G.evaluate(v3);
7     return [[KEM_PQ.DeriveKeyPair(v2[0 : kem_pq_nseed])[0], KEM_T.DeriveKeyPair(v2[kem_pq_n:
8   }
9   Bool Challenge([KEMPQCiphertextSpace, KEMTCiphertextSpace] ct0, [KEMPQCiphertextSpace, KEM
10  return false;
11 }
12 }
13 }
```

Challenger

```
1 Game Unbreakable() {
2   Bool Challenge(BitString<kem_pq_nss + kem_t_nss + kem_pq_nct + kem_t_nct + kem_pq_nek + ke
3   return false;
4 }
5 }
6 }
```

GAME HOPS

- LEAK\_BIND\_K\_CT(hybrid).B...
- KDFCollisionResistance(...)
- KDFCollisionResistance(...)
- LEAK\_BIND\_K\_CT(hybrid).B...