UNIVERSITY OF WATERLOO

Proving a real-world thing with an LLM and formal verification

Douglas Stebila

University of Waterloo

Based on various joint work with Ross Evans, Matthew McKague, and Camryn Steckel.

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

https://prooffrog.github.io/
https://www.douglas.stebila.ca/research/presentations/

Cryptographic result

draft-irtf-cfrg-hybrid-kems (Connolly, Barnes, Grubbs)

	Traditional part is a group	Traditional part is a KEM
Universal combiner hash everything	**UG** $H(ss_{PQ}, ss_T, ct_{PQ}, ct_T, ek_{PQ}, ek_T)$ like StarFortress [CG26]	**UK** $H(ss_{PQ}, ss_T, ct_{PQ}, ct_T, ek_{PQ}, ek_T)$ like [GHP18]
C2PRI combiner PQ ciphertext and key not hashed; needs C2PRI of the PQ KEM	**CG** $H(ss_{PQ}, ss_T, ct_T, ek_T)$ like X-Wing [BCD+24]	**CK** $H(ss_{PQ}, ss_T, ct_T, ek_T)$ like StarHunters [COSS26]

$\times 2$: **seeded form** (secret key is seed)
$+$ **expanded form** (secret key is expanded mathematical value).

[BCD+24] X-Wing, IACR CiC 2024. [CG26] StarFortress, IACR CiC 2026. [COSS26] StarHunters, ePrint 2026/427. [GHP18] KEM combiners, PKC 2018.

Target security properties

	UG		UK		CG		CK	
	seeded	expanded	seeded	expanded	seeded	expanded	seeded	expanded
IND-CCA (PQ branch)	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper
IND-CCA (T branch)	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper
LEAK-BIND-K-CT	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch
LEAK-BIND-K-PK	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch

paper — hand-written proof, mostly peer-reviewed [BCD+24] [CG26] [GHP18] [COSS26]

sketch — informal argument in the draft

claim by PRG — claimed to follow from the expanded form “by PRG security” (draft -12, Section 6.4.1

HON and MAL binding not targeted by the draft. Binding notions from [CDM24] Cremers, Dax, Medinger, ACM CCS 2024.

Our results: 57 theorems

	UG		UK		CG		CK	
	seeded	expanded	seeded	expanded	seeded	expanded	seeded	expanded
Correctness	✓	✓	✓	✓	✓	✓	✓	✓
IND-CCA (PQ branch)	✓	✓	✓	✓	✓	✓	✓	✓
IND-CCA (T branch)	✓ ROM (KDF)	✓ ROM (KDF)	✓	✓	✓ ROM (KDF)	✓ ROM (KDF)	✓	✓
LEAK-BIND-K-CT	✓	✓	✓	✓	✓ ROM (PRG)	✓	✓ ROM (PRG)	✓
LEAK-BIND-K-PK	✓	✓	✓	✓	✓ ROM (PRG)	✓	✓ ROM (PRG)	✓
HON-BIND-K-CT	implied	implied	implied	implied	✓ std	implied	✓ std	implied
HON-BIND-K-PK	implied	implied	implied	implied	✓ std	implied	✓ std	implied

Legend: ✓ standard model ✓ ROM (KDF) combiner KDF modelled as a RO implied by LEAK

✓ ROM (PRG) proved with the seed-expansion PRG modelled as a RO, not in the standard model

Provable security, computational model. Advantage bounds but not runtime analysis.

All checked in ProofFrog; some re-checked in EasyCrypt.

Most of the draft's claims hold, but a few surprises. No attacks identified.

Surprise: LEAK binding for seeded CG and CK

LEAK-BIND gives the adversary the decapsulation key.

In seeded form, the key is the seed.

C2PRI combiners CG and CK

- The KDF input omits ct_{PQ} and ek_{PQ}, so hybrid binding has to come from binding of the PQ KEM.
- The reduction gets the PQ key pair from its challenger, but must give the adversary one hybrid seed that expands to it.
- Standard-model PRG security doesn't help: the adversary holds the seed and can recompute the PRG.

We were able to show

- LEAK-BIND in the ROM, by programming the PRG
- HON-BIND in the standard model
  (adversary gets only a decapsulation oracle)

I don't have an attack, but also don't have an impossibility result.

Technical findings

1. LEAK binding for seeded CG and CK proved only with the PRG as a random oracle.
2. IND-CCA (T) of UG and CG need strong DH over shared-secret encodings, not group elements. (Becomes an issue with x-coordinate encoding.)
3. Seeded analyses assume DeriveKeyPair on a uniform seed is distributed like GenerateKeyPair. True of ML-KEM; but not stated explicitly in the draft's generic formulation.
4. Draft -12 allows for explicit rejection in some parts, but isn't fully addressed in the document.
5. KDF indifferentiability is needed only for the traditional branch of UG and CG. A PRF suffices elsewhere; binding needs only collision resistance.

CFRG chairs' response

"The additional machine-assisted analysis submitted by Douglas Stebila and Camryn Steckel reports no attack, supports most of the draft's security claims, and supports publication after corrections to the seeded CG/CK binding claims.
That analysis is useful evidence but is not treated as dispositive merely because some proofs were machine checked. Its authors expressly identify limitations: ProofFrog and its EasyCrypt exporter are research prototypes; the exported models have not received expert EasyCrypt review; explicit rejection and concrete instantiations were not modeled; classical random-oracle proofs do not by themselves establish QROM results; and LLM assistance was used extensively in the tooling, models, proof development, and report. The chairs therefore checked the findings against the draft text and underlying security arguments."

https://mailarchive.ietf.org/arch/msg/cfrg/v9fEHQj3QTUpdu72AzjyyrY4j2g/

Proving it with an LLM and formal verification

Toolchain

ProofFrog prooffrog.github.io

- Focused on game-hopping proofs.
- DSL for primitives, schemes, games, proofs. A property is a pair of games à la *Joy of Cryptography*.
- Checks a hop by canonicalizing both games' code and comparing.

Limitations: new, untested, under development. Large trusted base, partly agent-written. Far narrower and far less assurance than EasyCrypt. At best "moderate" assurance.

Claude Code

- Writes the ProofFrog files: schemes, games, intermediate games, reductions.
- Runs the engine on them through an MCP server.
- When a hop fails, it can:
  - Inspect intermediate proof state.
  - Revise proof steps.
  - Propose engine revisions.

Evans, McKague, Stebila. ProofFrog: A tool for verifying game-hopping proofs. ePrint 2025/418.

Division of labour

Me

- Set the goals: every variant × every security property.
- Read and edited every primitive, scheme and security definition.
- Supplied simpler prototype proofs to imitate (basic hybrid, GHP18, StarFortress).
- Decided what is an acceptable assumption (KeyGenEquiv; strong DH over encodings).
- Figured out path through LEAK-BIND ROM / std model issues.
- Read every theorem statement and assumption list. Identified limitations.
- Edited the CFRG report.

Claude

- Read the Internet-Draft; drafted the schemes and games.
- Wrote most of the 57 proofs (game hops/reductions) with no proof-strategy guidance for most.
- Extended the engine when a proof needed a missing transform.
- Drafted a report for the CFRG. 😔
- Wrote the EasyCrypt exporter, and the EasyCrypt helper proofs directly.

Time: about 40 hours of human effort, May to July. ish: intertwined with engine development. The EasyCrypt export has taken far longer and is ongoing.

What could go wrong?

Failure	Caught by	In this project
A hop is not a valid game transition	The checker	Happens lots during proof development; none by the end
The checker is unsound	Unchecked, for ProofFrog. Some reassurance if proofs are exported to a more trustworthy tool.	EasyCrypt-accepting proofs for many but not all claims
The proof verifies, but of a weaker theorem (an added assumption, a weakened game)	A human reading the theorem statement and its assumption list	I did observe Claude doing this sometimes.
The scheme or security properties do not match the standard	A human reading the scheme and games against the text	CFRG evaluation of report
The security definitions are wrong	Peer review of the definitions, separately from this work	Reused CDM24 and GHP18 rather than inventing notions

LLM strengths and weaknesses

Claude was good at:

- Grinding through 40 structurally similar proofs
- Extracting a proof strategy from a PDF (StarFortress)
- Extending the engine when a transform was missing
- Pushing through formal verification syntax and proof tactics

Where I needed to step in:

- Diagnosing why the seeded LEAK-BIND proofs failed, and choosing the way out (ROM for LEAK, standard model for HON)
- Checking the LLM didn't "fix" a failing proof by weakening the theorem
- Deciding what assumptions could be introduced
- Evaluating technical findings and limitations

Biggest concern: a proof that verifies of an unintended theorem.

Reflections

LLMs + formal verification enabled exhaustive analysis

By hand:

	UG		UK		CG		CK	
	seeded	expanded	seeded	expanded	seeded	expanded	seeded	expanded
IND-CCA (PQ branch)	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper
IND-CCA (T branch)	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper	claim by PRG	paper
LEAK-BIND-K-CT	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch
LEAK-BIND-K-PK	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch	claim by PRG	sketch

ProofFrog + Claude:

	UG		UK		CG		CK	
	seeded	expanded	seeded	expanded	seeded	expanded	seeded	expanded
Correctness	✓	✓	✓	✓	✓	✓	✓	✓
IND-CCA (PQ branch)	✓	✓	✓	✓	✓	✓	✓	✓
IND-CCA (T branch)	✓ ROM (KDF)	✓ ROM (KDF)	✓	✓	✓ ROM (KDF)	✓ ROM (KDF)	✓	✓
LEAK-BIND-K-CT	✓	✓	✓	✓	✓ ROM (PRG)	✓	✓ ROM (PRG)	✓
LEAK-BIND-K-PK	✓	✓	✓	✓	✓ ROM (PRG)	✓	✓ ROM (PRG)	✓
HON-BIND-K-CT	implied	implied	implied	implied	✓ std	implied	✓ std	implied
HON-BIND-K-PK	implied	implied	implied	implied	✓ std	implied	✓ std	implied

When I started this, I didn't expect I would find anything interesting.

Surprise was in a combination that a paper hadn't been written about, and maybe no one would have bothered to write a paper about.

I'm not sure I would have had the patience to prove all 57 by hand.

When a proof costs an order of magnitude less, we can afford to do more of them.

Formal verification probably mattered: prevented an LLM from writing a plausible proof sketch paragraph agreeing with the draft.

Although I still wasn't fully exhaustive: another $\times 2$ for explicit rejection that I haven't done.

What does this mean for training students?

I would have given (a subset of this) to a student for a Master's thesis or first PhD paper.

- Model a relatively straightforward standard. Prove its claimed properties. Write it up.

It is now about 40 hours with a coding agent.

What do we give students now? How do they get started?

Provable security for practitioners?

We're now in a world where a standards editor or an implementer can get moderate-assurance proofs of their own construction in a weekend.

The LLM didn't...

- Define the security notions (CDM24, GHP18)
- Choose what to prove
- Know which assumptions are acceptable
- Build a trustworthy checker
- Interpret the result for the standard

Candidate roles for the research community

- Definitions and models
- The formal verification checkers and their trust
- Reviewing models and assumptions rather than proofs
- Other characteristics: concrete instantiations, QROM, implementations, side channels
- Deciding what counts as a result

Proving a real-world thing with an LLM and formal verification

Douglas Stebila, University of Waterloo

ProofFrog:

`prooffrog.github.io`

CFRG hybrid KEMs analysis:

`github.com/ProofFrog/examples`

Slides:

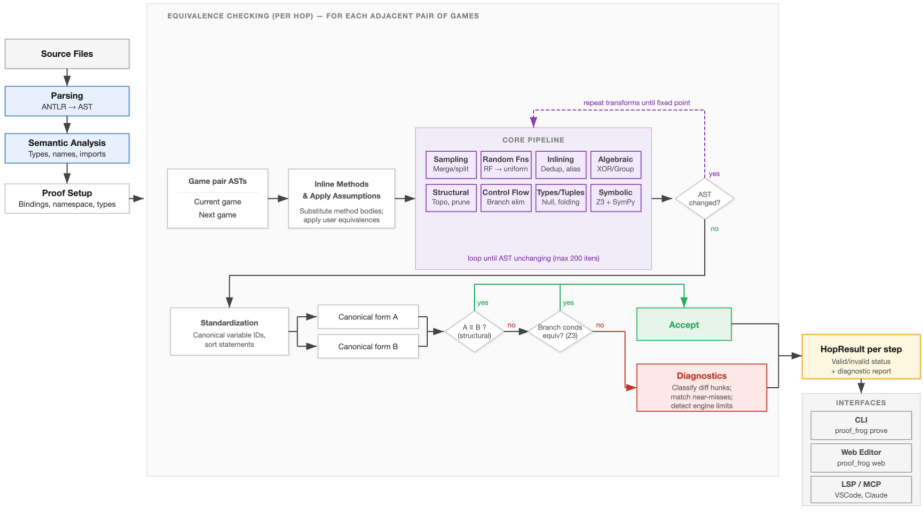
`www.douglas.stebila.ca/research/presentations/`

Questions for the rest of the session

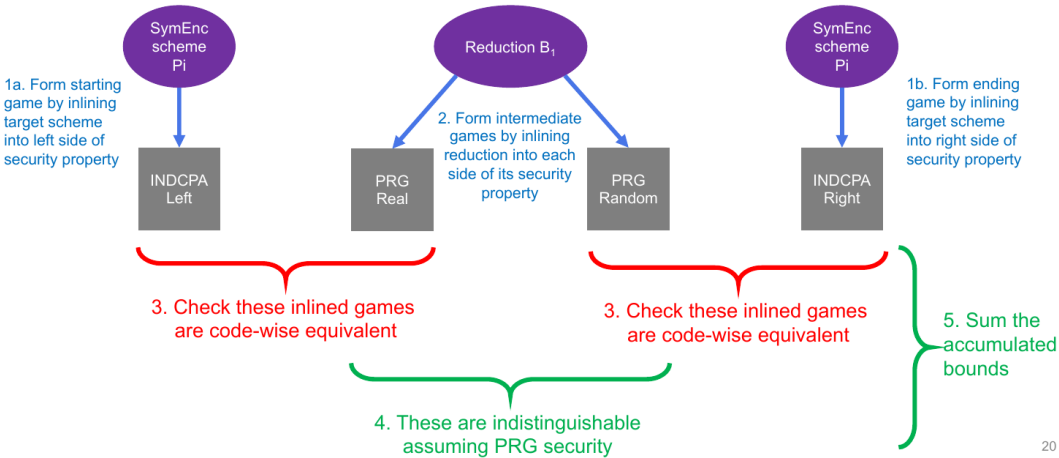
1. What is necessary (technically, presentation-wise) to communicate/trust/use/publish a result like this?
2. What is the role of the applied cryptography research community as protocol analysis changes?

Appendix

ProofFrog engine



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

A security definition in ProofFrog: LEAK-BIND-K-PK

```
Game Breakable(KEM K) {
    K.EncapsKey ek0;   K.EncapsKey ek1;
    K.DecapsKey dk0;   K.DecapsKey dk1;

    [K.EncapsKey, K.DecapsKey,
     K.EncapsKey, K.DecapsKey] Initialize() {
        [ek0, dk0] = K.KeyGen();
        [ek1, dk1] = K.KeyGen();
        return [ek0, dk0, ek1, dk1];
    }

    Bool Challenge(K.Ciphertext ct0, K.Ciphertext ct1) {
        K.SharedSecret ss0 = K.Decaps(dk0, ct0);
        K.SharedSecret ss1 = K.Decaps(dk1, ct1);
        return ss0 == ss1 && ek0 != ek1;
    }
}

Game Unbreakable(KEM K) {
    ... identical, except ...
    Bool Challenge(K.Ciphertext ct0, K.Ciphertext ct1) {
        ...
        return false;
    }
}

export as LEAK_BIND_K_PK;
```

- A property is a pair of games; the advantage is the distinguishing advantage between them.
- `Initialize` runs once; its output is the adversary's input. Here: two honest key pairs, including both decapsulation keys. That is the "LEAK".
- `Challenge` is an oracle. The win condition lives in the difference between the two bodies.
- Decapsulation is total here: no rejection symbol. Sound for implicitly rejecting KEMs such as ML-KEM; not for groups such as P-256 or ristretto255, which can reject.

Report Appendix A;
games/KEM/Binding/LEAK_BIND_K_PK.game in the examples repository.

What a theorem looks like: one binding bound

CG seeded, LEAK-BIND-K-PK, seed-expansion PRG modelled as a random oracle

$$\mathrm{Adv}^{\text{LEAK-BIND-K-PK}}_{\text{CG-seeded}}(A) \le \mathrm{Adv}^{\text{LEAK-BIND-K-PK}}_{\mathrm{KEM}_{PQ}}(B_1) + \mathrm{Adv}^{\mathrm{CR}}_{\mathrm{KDF}}(B_2) + 2 \cdot \mathrm{Adv}^{\mathrm{KeyGenEquiv}}_{\mathrm{KEM}_{PQ}}(B_3) + 2^{1-\lambda}$$

- Every proof file declares its bound; the engine synthesizes one from the hop sequence and checks the claim against it.
- Binding for the universal combiners reduces to collision resistance of the KDF alone. Binding for the C2PRI combiners additionally needs the same binding notion of the PQ component.
- No binding result needs indifferentiability of the KDF nor C2PRI.
- The full ledger of assumptions and coefficients per theorem is in `BOUNDS-CFRG-20260722.md`.

A proof file: CG seeded, LEAK-BIND-K-PK in the ROM

```
assume:
    LEAK_BIND_K_PK(KEM_PQ);
    KDFCollisionResistance(H);
    KeyGenEquiv(KEM_PQ);
    CGLazyROTwoSeeded(KEM_PQ, NG, lambda);

theorem:
    LEAK_BIND_K_PK_ROM(hybrid, BitString<lambda>,
        BitString<KEM_PQ.Nseed + NG.Nseed>, G_RO);

bound:
      advantage(KeyGenEquiv(KEM_PQ) compose R_KG_L)
    + advantage(LEAK_BIND_K_PK(KEM_PQ) compose R_PQ_Bind)
    + advantage(KDFCollisionResistance(H) compose R_KDF)
    + advantage(KeyGenEquiv(KEM_PQ) compose R_KG_R)
    + 2 ^ (1 - lambda);

games:
    LEAK_BIND_K_PK_ROM(hybrid, ...).Breakable
        against LEAK_BIND_K_PK_ROM(hybrid, ...).Adversary;
    CGLazyROTwoSeeded(KEM_PQ, NG, lambda).Honest
        compose R_LazyRO_L(...) against ...;
    ...   (14 games, 13 hops)

// then the six reductions R_LazyRO_L, R_KG_L, R_PQ_Bind,
// R_KDF, R_KG_R, R_LazyRO_R, written as programs
```

What `proof_frog prove` does with it:

- 13 hops: 7 code equivalences settled by canonicalization, 6 assumption hops.
- The two `CGLazyROTwoSeeded` hops are the statistical fact that lets the PRG be programmed at two fresh seeds; collision term $2^{1-\lambda}$.
- The two `KeyGenEquiv` hops bridge DeriveKeyPair on a uniform seed and KeyGen. (Finding F3.)
- Synthesizes the advantage bound from the hop sequence and checks the claimed `bound:` against it.

Output ends with “Claimed bound verified” and “Proof Succeeded!”. The file is about 500 lines, most of it the six reductions.

Findings in full

	Finding	Draft sections	Chairs
F1	LEAK binding for CG-seeded and CK-seeded is established only in the ROM. The draft's preferred key format is the one with the weaker result.	5.2, 6.4.1, 6.4.2	Mandatory 1
F2	The traditional branches of UG and CG reduce to strong DH over the output of ElementToSharedSecret, not over group elements. The X-coordinate map is two-to-one, so neither assumption is known to imply the other.	4.2, 6.1.3, 6.2.1	Mandatory 2
F3	Every seed-based analysis assumes DeriveKeyPair on a uniform seed has the same distribution as GenerateKeyPair. True for ML-KEM by definition; not implied by the interface.	4.1, 5.2, 6.4.1	Mandatory 3
F4	Draft -12 allows component decapsulation to fail, but the four Decaps routines proceed as if it never does, and the CG/CK binding argument is not rejection-aware.	4.1, 4.2, 5.3–5.6, 6.4.2	Mandatory 4
F5	KDF indifferentiability is needed only for the UG and CG traditional branches. Six of eight IND-CCA results need only a PRF; every binding result needs only collision resistance.	6.1.5, 6.2.1	Useful, not a blocker

ProofFrog → EasyCrypt export

Design: per transform, not per hop

- A ProofFrog hop is a chain of canonicalization transforms applied to both games until they agree.
- The exporter emits one EasyCrypt micro-lemma per transform application and glues the chain by transitivity.
- Each micro-lemma is closed by a static canned tactic, a synthesized parametric tactic, a tactic cached from an interactive session, or `admit`.
- An accepted micro-lemma is a machine-checked test case for one application of one transform: evidence about the engine, not a proof of it.
- MCP servers for ProofFrog and EasyCrypt let Claude step proofs interactively.

Status

- Exporter written entirely by LLM.
- At the report, 2026-07-22: 28 of 57 CFRG proofs accepted admit-free. Eventually rose to 51/57.
- Refactor in process to change exporter architecture; currently at 17/57; work in progress.

Limitations of the analysis (report Section 5)

Not modelled or not checked

- Concrete instantiations: nothing about ML-KEM, X25519 or P-256 beyond the properties assumed of them.
- Quantum adversaries: standard-model results carry over if the assumptions do; ROM results do not.
- Explicit rejection: decapsulation is total in our models. Sound for ML-KEM and X25519; binding results do not cover P-256 or ristretto255, whose decoding can fail.
- MAL-BIND: neither the draft nor we prove it.
- Byte-level encodings and labels are abstracted; injectivity is assumed.
- Models built against draft -10; differences to -12 tabulated in the report.

Assurance

- ProofFrog is a research prototype. The canonicalization pipeline and the FrogLang semantics it assumes are under active development. A verified hop is evidence, not a formal proof.
- EasyCrypt exports are accepted as written, but the modules, adversary classes, memory restrictions, axioms and distributions are ours and have not been reviewed by an EasyCrypt expert.
- LLM assistance throughout: models, proofs, engine, exporter, report. We read every primitive, scheme, definition, theorem statement and assumption list ourselves.
- The proof files are human-readable and name every reduction; a reader can evaluate them as they would a pen-and-paper proof.

ProofFrog limitations

- Automation fails $\Rightarrow$ no manual recourse
- Immature: new proofs often require engine extensions
- Limited expressivity, in proof techniques and in mathematical constructions
- Limited mathematical and probabilistic reasoning
- No verification of the engine itself; the transforms are the trusted base
- Not yet peer reviewed
- Generative AI used for engine development, so the checker is partly agent-authored, like the proofs it checks
- Adversaries are classical; ROM results not lifted to the QROM