Robust Logical Foundations for Mechanizing Post-Quantum Cryptography in Squirrel

Annual meeting of the GT-MFS

Adrien Koutsos Inria Paris

Joint work with:

David Baelde Univ Rennes, IRISA, CNRS

Antoine Dallon AMIAD

Stéphanie Delaune Univ Rennes, IRISA, CNRS

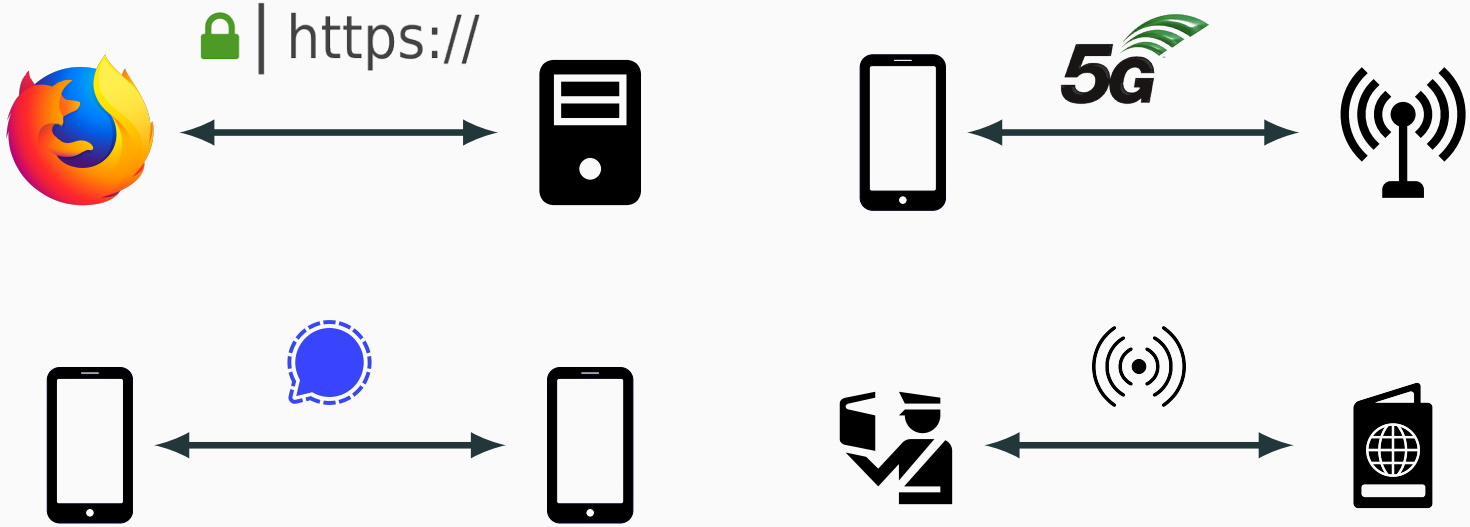
Charlie Jacomme Inria Nancy

March 26, 2026, Luz-Saint-Sauveur

Context

Context

Cryptography is pervasive

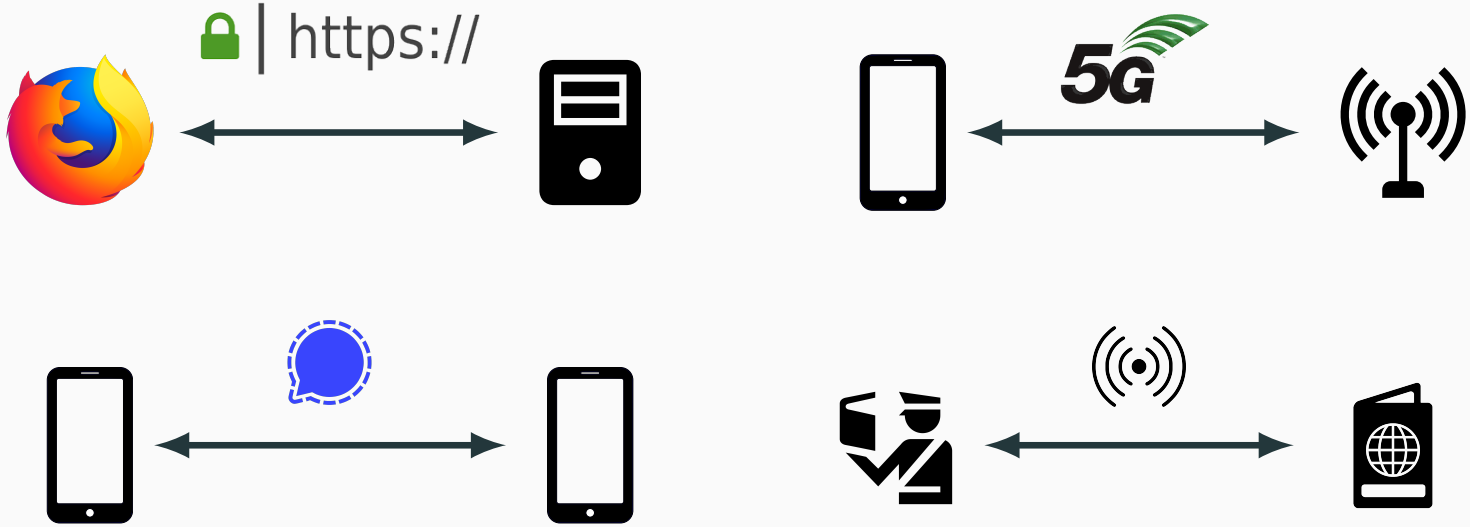


- **Security properties:** authentication, confidentiality, privacy, ...
- **Attacks** can be very **damageable**: privacy breach, theft, ...

⇒ Ensure that **cryptographic designs are secure**:
high-level specification, implementation, ...

Context

Cryptography is pervasive



- **Security properties:** authentication, confidentiality, privacy, ...
- **Attacks** can be very **damageable**: privacy breach, theft, ...

⇒ Ensure that **cryptographic designs are secure**:
high-level specification, implementation, ...

Context: Provable Security

Provable security

Mathematical proofs of security

$$\forall \text{😈} \in \mathcal{C}.\ \Pr\left(\text{😈 breaks } \mathcal{P}\right) \leq \epsilon$$

Context: Provable Security

Provable security

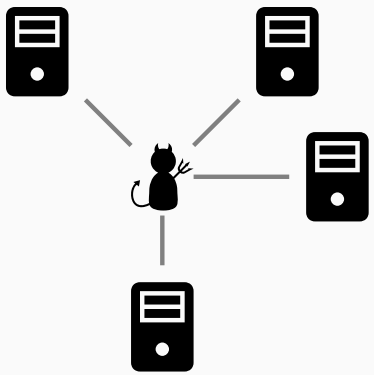
Mathematical proofs of security

$$\forall \text{😈} \in \mathcal{C}.\ \Pr\left(\text{😈 breaks } \mathcal{P}\right) \leq \epsilon$$

Attacker model: capture realistic attack scenario

Network capabilities

Adversary controls the networks



Computational capabilities

$\mathcal{C} = \text{PPTM}$

- Polynomial-time
- Probabilistic
- Turing Machine

Context: Provable Security

Proof techniques for provable security

$$\forall \text{😈} \in \mathcal{C}.\ \Pr\left(\text{😈 breaks } \mathcal{P}\right) \leq \epsilon$$

Context: Provable Security

Proof techniques for provable security

$$\forall \text{😺} \in \mathcal{C}.\ \Pr\left(\text{😺 breaks } \mathcal{P}\right) \leq \epsilon$$

- **Information-theoretic** arguments

$$\Pr\left(k \xleftarrow{\$} \{0,1\}^\eta : \text{😺}() = k\right) = \frac{1}{2^\eta}$$

- **Computational** arguments such as **cryptographic reductions**

  Similar to theoretical complexity reductions (e.g. Hamilton $\Rightarrow$ SAT)

$$\left(\exists \text{😺} \in \mathcal{C} \text{ breaking } \mathcal{P}\right) \Rightarrow \left(\exists \text{😈} \in \mathcal{C} \text{ breaking } \mathcal{H}\right)$$

  Hardness assumption $\mathcal{H}$ cannot be broken by any 😈 $\in \mathcal{C}$

  Examples: Discrete-Log, IND-CPA, ...

Context

The problem with provable security

- Security proofs are **intricate**.

  (probabilities, complexity, concurrency)
- Cryptographic designs are **critical** systems.

$\Rightarrow$ Errors are **common-place** and **costly**.

Context

The problem with provable security

- Security proofs are **intricate**.

  (probabilities, complexity, concurrency)
- Cryptographic designs are **critical** systems.

$\Rightarrow$ Errors are **common-place** and **costly**.

Computer-Aided Cryptography (CAC)

Mechanization: machined-checked cryptographic proofs

$\Rightarrow$ High level of confidence.

CAC frameworks: CryptoVerif, Squirrel, EasyCrypt, SSProve

$$\models \quad \left(\forall \text{😈} \in \mathcal{C}.\ \Pr\left(\text{😈 breaks } \mathcal{P}\right) \leq \epsilon\right) \quad \checkmark$$

Context

The problem with provable security

- Security proofs are **intricate**.

  (probabilities, complexity, concurrency)
- Cryptographic designs are **critical** systems.

$\Rightarrow$ Errors are **common-place** and **costly**.

Computer-Aided Cryptography (CAC)

Mechanization: machined-checked cryptographic proofs

$\Rightarrow$ High level of confidence.

CAC frameworks: CryptoVerif, Squirrel, EasyCrypt, SSProve

$$\models \quad \left(\forall \text{😈} \in \mathcal{C}.\ \Pr\left(\text{😈 breaks } \mathcal{P}\right) \leq \epsilon\right) \quad \checkmark$$

Are we done?

Context: Quantum Computers

Attacker model: capture realistic attack scenario

Context: Quantum Computers

Attacker model: capture realistic attack scenario

Quantum Computers

- Working quantum computer (QC) may arrive
- QC breaks many existing crypto systems

  Discrete logarithm, Diffie-Hellman, ...

Context: Quantum Computers

Attacker model: capture realistic attack scenario

Quantum Computers

- Working quantum computer (QC) may arrive
- QC breaks many existing crypto systems
  Discrete logarithm, Diffie-Hellman, ...

Post-Quantum Cryptography (PQC)

Secure cryptography against quantum adversaries.

- adversary: **quantum**
- protocol: **classical**

Context: Post-Quantum Cryptography

PQC effort in progress

- PQ primitives: ML-KEM, ML-DSA
- PQ protocols: Signal (PQXDH, SPQR), iMessage (PQ3)

Context: Post-Quantum Cryptography

PQC effort in progress

- PQ primitives: ML-KEM, ML-DSA
- PQ protocols: Signal (PQXDH, SPQR), iMessage (PQ3)

Provable PQC

$$\forall \text{😼} \in \mathcal{C}.\ \Pr(\text{😼 breaks } \mathcal{P}) \leq \epsilon \qquad (\mathcal{C} = \mathsf{PQTM})$$

CAC for PQC *(work-in-progress)*

Mechanized cryptographic proofs of PQ security.

PQC Frameworks: CryptoVerif, Squirrel, EasyPQC, qrhl-tool

The Squirrel Prover: Theoretical Foundations

Theoretical foundations: the **CCSA logic**

$$(\text{cat} \mid \mathcal{P}) \quad \models \quad \Phi$$

1. Modeling

- **Language:** pure λ-calculus

  Execution model: encode $(\text{cat} \mid \mathcal{P})$ interactions
- **FO formulas** for **asymptotic cryptography**
  - **Reachability**: $[\phi_{\mathcal{P}}]$
  - **Indistinguishability:** $\vec{u}_{\mathcal{P}} \sim_c \vec{u}_{\mathcal{P}'}$

2. Reasoning

- **Reasoning rules** valid w.r.t. classical attackers.

The Squirrel Prover: The Tool

Interactive proof assistant

Users prove goals using tactics.

- **Generic maths**, e.g. apply, rewrite, smt.
- **Crypto**, e.g. fresh, deduce, crypto.
  Automated **reduction synthesis** procedures.
  $(\exists$  $\in \mathcal{C}. \ldots) \Rightarrow (\exists$  $\in \mathcal{C}. \ldots)$

Development done in Proof-General.

As in Rocq, EasyCrypt ...

Open-source: `https://squirrel-prover.github.io/`

Context: PQ-Squirrel

Limitations of PQ-Squirrel

- **Expressivity:**

  Capture PQTMs using **black-box interactive machines**

  $\Rightarrow$ quantum values not represented (e.g. no QROM)

- **Maintainability:**
  - Unusual **stateful** semantics
  - Distance with implem.
  - Lacks latest improvements (theory, implem.), e.g. crypto, smt

Context: PQ-Squirrel

Limitations of PQ-Squirrel

- **Expressivity:**
  Capture PQTMs using **black-box interactive machines**
  $\Rightarrow$ quantum values not represented (e.g. no QROM)
- **Maintainability:**
  - Unusual **stateful** semantics
  - Distance with implem.
  - Lacks latest improvements (theory, implem.), e.g. crypto, smt

Goal

Design a **robust** PQ extension of Squirrel

Context: Building a PQC Verification Framework

Roadmap to adapt a CAC framework to PQC.

- **Modeling:** capture quantum computations and adversaries
- **Reasoning:** capture PQ cryptographic arguments

Context: Building a PQC Verification Framework

Roadmap to adapt a CAC framework to PQC.

- **Modeling:** capture quantum computations and adversaries
- **Reasoning:** capture PQ cryptographic arguments

Challenge **(Reasoning)**

$$(\exists \text{😺} \in \mathcal{C}.\ \text{breaking } \mathcal{P}) \Rightarrow (\exists \text{😈} \in \mathcal{C}.\ \text{breaking } \mathcal{H})$$

Reductionistic arguments must be adapted:

- Exclude **insecure assumptions**, e.g. Discrete-Log, DDH.
- **No-cloning theorem:** ensure that 😈 is a PQTMs

Contributions

- **Execution model** for PQC in Squirrel
- **Faithful logic** for PQC
- **Adapt Squirrel proof systems**
  - Support latest features, e.g. smt, crypto
- **Implementation**
- **Validation** through **case-studies**
  - Hybrid KEM Combiners
  - Hybrid Key-Exchanges

An PQ Execution Model

Execution Model

Goal: encode $(\text{😈} \mid \mathcal{P})$ interactions as Squirrel terms

- **Existing encoding** for classical 😈 **unsuitable**
  Require state re-computations $\Rightarrow$ violates no-cloning
- We need a **new execution model for PQC**

Execution Model: Squirrel Primer

Squirrel Language

Pure λ-calculus with **early-sampled randomness**

Pure language

Functional encoding with **explicit state**:

stateful	$\Rightarrow$	**att** stateless
$(\text{out} \leftarrow \text{in}); p)$	$\Rightarrow$	**let** $(\text{out}, \text{st}') = \textbf{att}(\text{in}, \text{st})$ **in** $\tilde{p}$

Execution Model: Squirrel Primer

Squirrel Language

Pure λ-calculus with **early-sampled randomness**

Pure language

Functional encoding with **explicit state**:

 stateful $\Rightarrow$ **att** stateless

$(\text{out} \leftarrow$  $(\text{in}); p)$ $\Rightarrow$ **let** $(\text{out}, \text{st}') = \textbf{att}(\text{in}, \text{st})$ **in** $\tilde{p}$

Early-sampled randomness

Names = arrays of pre-sampled i.i.d. randomness

$$\text{n} : \text{timestamp} \to \{0,1\}^\eta$$

$$\begin{array}{l} \text{x} \xleftarrow{\$} \{0,1\}^\eta; \\ \text{y} \xleftarrow{\$} \{0,1\}^\eta; p \end{array} \Rightarrow \begin{array}{l} \textbf{let } \text{x} = \text{n}\ (t) \quad \textbf{in} \\ \textbf{let } \text{y} = \text{n}\ (t+1)\ \textbf{in } \tilde{p} \end{array}$$

Execution Model: QC Primer

Classical deterministic machine: bit-string


$_{\mathsf{d}}(\mathsf{in}) = v \qquad\qquad v \in \{0,1\}^*$

Execution Model: QC Primer

Classical deterministic machine: bit-string

$$\text{😈}_{\mathsf{d}}(\mathsf{in}) = v \qquad\qquad v \in \{0,1\}^*$$

Classical probabilistic machine: bit-string distribution

$$\text{😈}_{\mathsf{p}}(\mathsf{in}) = \sum_{v \in \{0,1\}^*} p_v \cdot v \qquad\qquad \sum_v p_v = 1, \quad \forall v.\, p_v \in \mathbb{R}^+$$

Example: $\frac{1}{2}$ `"head"` $+ \frac{1}{2}$ `"tail"`

Execution Model: QC Primer

Classical deterministic machine: bit-string

$$😈_{\mathsf{d}}(\mathsf{in}) = v \qquad\qquad v \in \{0,1\}^*$$

Classical probabilistic machine: bit-string distribution

$$😈_{\mathsf{p}}(\mathsf{in}) = \sum_{v \in \{0,1\}^*} p_v \cdot v \qquad\qquad \sum_v p_v = 1, \quad \forall v.\, p_v \in \mathbb{R}^+$$

Example: $\frac{1}{2}$ `"head"` $+ \frac{1}{2}$ `"tail"`

Quantum machine: bit-string superposition

$$😈_{\mathsf{q}}(\mathsf{in}) = \sum_{v \in \{0,1\}^*} q_v \cdot |v\rangle \qquad\qquad \sum_v |q_v|^2 = 1, \quad \forall v.\, q_v \in \mathbb{C}$$

Example: $\frac{1}{\sqrt{2}} |$`"head"`$\rangle - \frac{1}{\sqrt{2}} |$`"tail"`$\rangle$

Execution Model: QC Primer

Measurement: bit-string superposition $\hookrightarrow$ bit-string distribution

$$\sum_{v \in \{0,1\}^*} q_v \cdot |v\rangle \xhookrightarrow[\text{measure}]{} \sum_{v \in \{0,1\}^*} |q_v|^2 \cdot v$$

Execution Model: QC Primer

Measurement: bit-string superposition $\hookrightarrow$ bit-string distribution

$$\sum_{v \in \{0,1\}^*} q_v \cdot |v\rangle \xhookrightarrow[\text{measure}]{} \sum_{v \in \{0,1\}^*} |q_v|^2 \cdot v$$

Partial measurement (first N bits):

$$\text{q}(\text{in}) \xhookrightarrow[\substack{\text{partial measure} \\ \text{(N bits)}}]{} \text{Distr}\left(\{0,1\}^N \times \mathcal{H}_{\{0,1\}^*} \right)$$

Execution Model: QC Primer

Measurement: bit-string superposition $\hookrightarrow$ bit-string distribution

$$\sum_{v \in \{0,1\}^*} q_v \cdot |v\rangle \xhookrightarrow[\text{measure}]{} \sum_{v \in \{0,1\}^*} |q_v|^2 \cdot v$$

Partial measurement (first N bits):

$$\text{🐱}_{\text{q}}(\text{in}) \xhookrightarrow[\substack{\text{partial measure} \\ \text{(N bits)}}]{} \text{Distr}\left(\{0,1\}^N \times \mathcal{H}_{\{0,1\}^*} \right)$$

Quantum adversary:

$\{0,1\}^N$: **classical output** $\mathcal{H}_{\{0,1\}^*}$: **quantum state**

Execution Model: Calling the Quantum Adversary

$$\text{😈}_{\text{q}}(\text{in}) \xhookrightarrow[\text{partial measure}]{} \text{Distr}\left(\{0,1\}^N \times \mathcal{H}_{\{0,1\}^*}\right)$$

Modeling a PQTM 😈$_{\text{q}}$ in Squirrel

- **Stateless attacker att** with **explicit state** st
- **Pre-sampled randomness** for measures:

$$\text{qrnd} : \text{timestamp} \to \text{qrand}$$

Encoding of 😈$_{\text{q}}(\text{in})$ for t-th call:

$$\textbf{let } (\text{out}, \text{st}') = \textbf{att}(\text{qrnd } t, \text{in}, \text{st}) \textbf{ in } \ldots$$

Execution Model: Protocol Interaction (Simplified)

Chaining 2 calls

$$
\begin{array}{l}
\mathsf{in} \leftarrow \overset{\text{q}}{\text{😈}}(\mathsf{out}); \\
\mathsf{out} \leftarrow \mathcal{P}(\mathsf{in}); \\
\mathsf{in} \leftarrow \overset{\text{q}}{\text{😈}}(\mathsf{out}); \\
\ldots
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcll}
\textbf{let}\ (\mathsf{in}, \mathsf{st}) & = & \textbf{att}(\mathsf{qrnd}\ t, & \mathsf{out}, \mathsf{st})\ \textbf{in} \\
\textbf{let}\ \mathsf{out} & = & \tilde{\mathcal{P}}(\mathsf{in}) & \textbf{in} \\
\textbf{let}\ (\mathsf{in}, \mathsf{st}) & = & \textbf{att}(\mathsf{qrnd}\ (t+1), & \mathsf{out}, \mathsf{st})\ \textbf{in} \\
\ldots & & &
\end{array}
$$

Execution Model: Protocol Interaction (Simplified)

Chaining 2 calls

$$
\begin{array}{l}
\mathsf{in} \leftarrow \text{😈}_{\mathsf{q}}(\mathsf{out}); \\
\mathsf{out} \leftarrow \mathcal{P}(\mathsf{in}); \\
\mathsf{in} \leftarrow \text{😈}_{\mathsf{q}}(\mathsf{out}); \\
\ldots
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcll}
\textbf{let}\ (\mathsf{in}, \mathsf{st}) & = & \textbf{att}(\mathsf{qrnd}\ t, & \mathsf{out}, \mathsf{st})\ \textbf{in} \\
\textbf{let}\ \mathsf{out} & = & \tilde{\mathcal{P}}(\mathsf{in}) & \textbf{in} \\
\textbf{let}\ (\mathsf{in}, \mathsf{st}) & = & \textbf{att}(\mathsf{qrnd}\ (t+1), & \mathsf{out}, \mathsf{st})\ \textbf{in} \\
\ldots & & &
\end{array}
$$

Chaining many calls: use **recursive** definitions

$$
\begin{array}{rcl}
\mathsf{in}\ (t+1) & = & \textbf{att}(\mathsf{qrnd}\ t, \mathsf{out}\ t, \mathsf{st}\ t)\#1 \\
\mathsf{st}\ (t+1) & = & \textbf{att}(\mathsf{qrnd}\ t, \mathsf{out}\ t, \mathsf{st}\ t)\#2 \\
\mathsf{out}\ t & = & \tilde{\mathcal{P}}(\mathsf{in}\ t) \\
\mathsf{frame}\ t & = & \langle \mathsf{out\ init}, \ldots, \mathsf{out}\ t \rangle
\end{array}
$$

Execution Model: Conclusion

Key ideas

- **Explicit state** (st t)
- **Measurement randomness** pre-sampled in qrnd
  - qrnd $\neq$ *program randomness*
  - capture a **physical phenomenon**

Careful, terms $\neq$ QTM

- Quantum values duplication

$$(\mathbf{att}(\mathsf{n}, 0), \mathbf{att}(\mathsf{n}, 0))$$

- Weirder, quantum randomness re-use

$$(\mathbf{att}(\mathsf{n}, 0), \mathbf{att}(\mathsf{n}, 1))$$

Still, all terms has a **well-defined semantics**.

A Faithful Logic for QC

A Faithful Logic for PQC

Cryptographic predicates

- **Reachability** $[\phi]$ e.g. $[😺(\text{frame } t) \neq \text{k}]$

$$\Pr(\textbf{not } [\![\phi]\!]) \leq \epsilon_{\text{negl}}$$

- **Indistinguishability** $u \sim v$ e.g. $\text{frame}_{\mathcal{P}}\ t \sim \text{frame}_{\mathcal{P}'}\ t$

  Classical $u \sim_c v$

$$\forall 😺 : \text{PPTM}.\ \ |\Pr(😺([\![u]\!]) = 1) - \Pr(😺([\![v]\!]) = 1)| \leq \epsilon_{\text{negl}}$$

  Quantum $u \sim_q v$

$$\forall 😺 : \text{PHTM}.\ \ |\Pr(😺([\![u]\!]) = 1) - \Pr(😺([\![v]\!]) = 1)| \leq \epsilon_{\text{negl}}$$

A Faithful Logic for PQC

Hybrid machine $\mathcal{C}$: PHTM

$$\mathcal{C}(\text{in}) = \text{fold}(\mathcal{C}_c, \mathcal{C}_q, \text{in})$$

- $\mathcal{C}_c$: PPTM, $\mathcal{C}_q$: PQTM
- Full computation in **polynomial-time**

Advantage

More **expressive**:

- $\mathcal{C}_c$ can have **classical oracles**
- $\mathcal{C}_q$ can have **quantum oracles**

A Faithful Logic for PQC

Early-sampled probabilities

Names: **finite** arrays of pre-sampled randomness.

$$\mathsf{n} : \mathsf{S} \to \mathsf{D}$$

Finiteness ensures that:

$$\Pr_{\rho}\left([\![u]\!](\rho) \in \mathsf{E}\right) \text{ well-defined for any E}$$

Otherwise, must ensure measurability of $([\![u]\!] \in E) \Rightarrow$ complex in general

Quantum measurement modeling

- (qrnd t) as large as we want
- **But** same size for all terms

$\Rightarrow$ **not always enough randomness!**

A Faithful Logic for PQC

Solution

- **Approximation models** $\mathbb{M}$:

  finite (qrnd t) $\qquad\qquad$ $\Pr([\![u]\!]_{\mathbb{M}} \in \mathsf{E})$ ✓

- **Exact models** $\mathbb{M}_{\mathsf{e}}$:

  infinite (qrnd t) $\qquad\qquad$ $\Pr([\![u]\!]_{\mathbb{M}_{\mathsf{e}}} \in \mathsf{E})$?

Adequacy Theorem

If u is **well-formed** and if (qrnd t) is **large enough**:

$$\mathsf{D}_{\mathsf{dist}}\left([\![u]\!]_{\mathbb{M}}, [\![u]\!]_{\mathbb{M}_{\mathsf{e}}}\right) \leq \epsilon_{\mathsf{negl}}$$

(Very roughly, well-formed = PQTM simulatable)

Outline

- **Execution model** for PQC in Squirrel ✓
- **Faithful logic** for PQC ✓
- **Adapt Squirrel proof systems**
- **Implementation** + **Case-studies**

Adapting Squirrel Proof System

Adapting the Proof System

Goal: adapt Squirrel reasoning capabilities to PQC

- **Core logical rules**: rewrite, apply, smt, ...
- **Cryptographic rules**:
  - Basic rules, e.g. fresh, fa, ...
  - Automated simplifications: deduce
  - Reductions to hardness assumptions: crypto

Adapting the Proof System: Core Logical Rules

- We use Squirrel existing semantics.
- $\sim_c$ and $\sim_q$ can **absorb a negligible error**.

$\Rightarrow$ **inherit non-reductionist rules for free.**

$$\text{rewrite}_c\ \frac{u' \sim_c v \qquad [u = u']}{u \sim_c v} \quad \Rightarrow \quad \text{rewrite}_q\ \frac{u' \sim_q v \qquad [u = u']}{u \sim_q v} \qquad \text{smt}\ \frac{\vdash_{\text{smt}} \phi}{[\phi]}$$

Adapting the Proof System

- **Core logical rules**: rewrite ✓, apply ✓, smt ✓, ...
- **Cryptographic rules**:
  - Basic rules, e.g. fresh ✓, fa, ...
  - Automated simplifications: deduce
  - Reductions to hardness assumptions: crypto

Difficulty

Remaining rules are **reduction-based**.

Reductionist Rules: Basic Rules

Classical rules

$$\mathsf{dup_c}\ \frac{u \sim_{\mathsf{c}} v}{u, u \sim_{\mathsf{c}} v, v} \qquad \mathsf{fa_c}\ \frac{u \sim_{\mathsf{c}} v \qquad f \in \mathsf{Lib}}{f(u) \sim_{\mathsf{c}} f(v)}$$

Issues

$$\frac{u_q \sim_{\mathsf{q}} v_q}{u_q, u_q \sim_{\mathsf{q}} v_q, v_q}$$

cloning quantum value ✗

$$\frac{\mathbf{att}(\mathsf{r}, u) \sim_{\mathsf{q}} v}{\mathbf{att}(\mathsf{r}, \mathbf{att}(\mathsf{r}, u)) \sim_{\mathsf{q}} \mathbf{att}(\mathsf{r}, v)}$$

quantum randomness r re-used ✗

Quantum rules

$$\mathsf{dup_q}\ \frac{u_c \sim_{\mathsf{c}} v_c}{u_c, u_c \sim_{\mathsf{c}} v_c, v_c} \qquad \mathsf{fa_q}\ \frac{u \sim_{\mathsf{q}} v \qquad \phi^{\mathsf{r}}_{\mathsf{fresh}}(u, v)}{\mathbf{att}(\mathsf{r}, u) \sim_{\mathsf{q}} \mathbf{att}(\mathsf{r}, v)}$$

Reductionist Rules: Bi-Deduction

- $\sim$ preserved by **public computation**
- **public computation** = **adversarially simulatable**
  $\Rightarrow$ complex because of **probabilistic dependencies**

Reductionist Rules: Bi-Deduction

- $\sim$ preserved by **public computation**
- **public computation** = **adversarially simulatable**
  $\Rightarrow$ complex because of **probabilistic dependencies**

Bi-deduction [CSF'22]

Simplifications of $\sim$ by **deterministic simulation**.

$$\mathsf{deduce}_{\mathsf{c}}\ \frac{u_0 \sim_{\mathsf{c}} u_1 \qquad \sharp(u_0, u_1) \rhd_{\mathsf{c}} \sharp(v_0, v_1)}{v_0 \sim_{\mathsf{c}} v_1}$$

- **Proof system for** $\rhd_{\mathsf{c}}$
- **Tactic** deduce: $-$ expressivity, $++$ automation

Reductionist Rules: Bi-Deduction

Quantum bi-deduction $\triangleright_q$

$\triangleright_c$: **deterministic** simulation

$\Downarrow$

$\triangleright_q$: **error-free** simulation

Error-free quantum machines PQTM_E

$$f(\text{in}) \xhookrightarrow[\text{partial measure}]{} \text{Dirac}\left(\{0,1\}^N \times \mathcal{H}_{\{0,1\}^*}\right) \approx \{0,1\}^N \times \mathcal{H}_{\{0,1\}^*}$$

Reductionist Rules: Bi-Deduction

Advantages of $\mathbf{PQTM_E}$

- No measurement randomness
  $\Rightarrow$ no probabilistic dependencies ✓
- Support basic manipulations of quantum values
  Example: swapping $(c, q) \rhd (q, c)$
- For more complex manipulations:
  automatic deduce + manual fa for $\mathbf{att}(\cdot)$

Reductionist Rules: Bi-Deduction

Proof system

$$\rhd_{\mathsf{q}} \quad \approx \quad \rhd_{\mathsf{c}} + \begin{array}{c}\textbf{linear usage}\\ \text{of quantum values}\end{array}$$

Example: transitivity

$$\frac{\mathbf{u} \rhd_{\mathsf{c}} \mathbf{w} \qquad \mathbf{u}, \mathbf{w} \rhd_{\mathsf{c}} \mathbf{v}}{\mathbf{u} \rhd_{\mathsf{c}} \mathbf{v}} \quad \Rightarrow \quad \frac{\mathbf{c}, \mathbf{q}_1 \rhd_{\mathsf{q}} \mathbf{w} \qquad \mathbf{c}, \mathbf{q}_2, \mathbf{w} \rhd_{\mathsf{q}} \mathbf{v}}{\mathbf{c}, \mathbf{q}_1, \mathbf{q}_2 \rhd_{\mathsf{q}} \mathbf{v}}$$

- **Classical** value $\mathbf{c}$ can be re-used
- **Quantum** value $\mathbf{q}_1$, $\mathbf{q}_2$ used **linearly**

Adapting the Proof System

- **Core logical rules**: rewrite ✓, apply ✓, smt ✓, ...
- **Cryptographic rules**:
  - Basic rules, e.g. fresh ✓, fa ✓, ...
  - Automated simplifications: deduce ✓
  - Reductions to hardness assumptions: crypto

Reductionist Rules: Cryptographic Bi-Deduction

Cryptographic reduction to game $\mathcal{G} = \sharp(\mathcal{G}_0, \mathcal{G}_1)$

$$v_0 \sim_c v_1 \quad \text{if} \quad \exists S : \text{PPTM}.\ S^{\mathcal{G}_0} = v_0 \ \wedge \ S^{\mathcal{G}_1} = v_1$$

Examples: IND-CCA, PRF, DDH

Classical cryptographic bi-deduction [CCS'24]

$$\cdots \vdash \sharp(u_0, u_1) \rhd_c^{\mathcal{G}} \sharp(v_0, v_1)$$

- **Complex semantics**: $\mathcal{S}$ probabilistic + $\mathcal{G}$ stateful ($\cdots$)
- **Proof system** for $\rhd_c^{\mathcal{G}}$
- **Automatic proof-search**, including induction
- **Tactic** crypto: + expressivity, + automation

Reductionist Rules: Cryptographic Bi-Deduction

Challenges

- $\mathcal{S}$ probabilistic $\Rightarrow$ quantum error-free **insufficient**
- Generalize $\mathcal{S}$ to PQTM **complex**:
  $\Rightarrow$ change semantics + rules + proof-search ✗

Key Idea

- **Encapsulate quantum manipulation** in the game
- **Safe quantum API** $\mathcal{Q}$: force **linear usage** of quantum state

Reductionist Rules: Cryptographic Bi-Deduction

Safe Quantum API

```
game 𝒬 = {
  (* quantum state *)
  var state : ℋ_message = ...;

  (* classical state, next protocol input *)
  var input : message = ...;

  (* update state using att *)
  oracle step (out) = {
    r ←$ qrand;
    (input,state) = att(r, (state,out));
  }

  (* retrieve last attacker input *)
  oracle get_input () = { return input; }
}
```

Reductionist Rules: Cryptographic Bi-Deduction

$$\exists\, \mathsf{A} : \mathsf{PPTM}\,.\, \mathsf{A}^{\mathcal{Q}\cdot\mathcal{G}} = u \qquad \Rightarrow \qquad \exists\, \mathsf{B} : \mathsf{PQTM}\,.\, \mathsf{B}^{\mathcal{G}} = u$$

$$\boxed{\mathsf{B}^{\mathcal{G}} = (\mathsf{A}^{\mathcal{Q}})^{\mathcal{G}} = \mathsf{A}^{\mathcal{Q}\cdot\mathcal{G}}}$$

- **Semantics** of $\rhd_{\mathsf{c}}^{\mathcal{G}}$ unchanged (except minor change for $\mathcal{G}$ quantum)
- **Proof system** unchanged
- Specialized **induction rule** for the quantum execution model

$$\frac{x \vdash \mathsf{in}\ x \rhd_{\mathsf{c}}^{\mathcal{G}}\ \mathsf{out}\ x}{t \rhd_{\mathsf{c}}^{\mathcal{G}\cdot\mathcal{Q}}\ \mathsf{frame}\ t}$$

  No manipulation of state by S.
- **Implementation**: re-use most machinery

Case-Studies

Case Studies

Case studies in our post-quantum Squirrel

- **KEM combiners (CPA/CCA)**:
  XOR, XOR-then-MAC, Dual-PRF, Nested Dual-PRF
- **Hybrid KEM-based KE protocols (strong secrecy)**:
  BCGNP [S&P'22], C_{Sigma}
- Use latest features: crypto, smt

Proof strategy

- First proof in classical setting ($\approx$ pers. **months**)
- Then, adapted to post-quantum ($\approx$ pers. **days**)

Conclusion

Conclusion

- **Execution model for PQC**
- **A faithful logic for PQC**

  Adequacy theorem
- **Adapt Squirrel proof systems**

  smt, crypto, . . .
- **Implementation** + **Case-studies**

  Hybrid KEM Combiners and KE

Conclusion

- **Execution model for PQC**
- **A faithful logic for PQC**

  Adequacy theorem
- **Adapt Squirrel proof systems**

  smt, crypto, . . .
- **Implementation** + **Case-studies**

  Hybrid KEM Combiners and KE

Thank you for your attention