Schroedinger’s Squirrel

Formal security proofs in a post-quantum world

Charlie Jacomme
CISPA Helmholtz Center for Information Security
November 19, 2021
Formal Methods for Security and Privacy
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

It’s my life!
The beautiful and frightening technological revolution
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services...

but our data is used against us!
The beautiful and frightening technological revolution

It's my life!

Sacrifice privacy in exchange of services...

but our data is used against us!
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

It's my life!

Sacrifice privacy in exchange of services... but our data is used against us!
The beautiful and frightening technological revolution

Sacrifice privacy in exchange for services...
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services...
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services...
The beautiful and frightening technological revolution

Sacrifice privacy in exchange of services... but our data is used against us!
Some people even need privacy to survive:

- Reporters in dangerous countries.
- Homosexual in countries where it is punished by law (still 69 in the world...).
- Uighurs tracked through their smartphones in China.

If we can’t have privacy, nobody can
Security and Privacy Matter!
Security and Privacy Matter!

We need:

- systems designed to provide security and privacy;
Security and Privacy Matter!

We need:

- systems designed to provide security and privacy;
- with guarantees that they do;
Security and Privacy Matter!

We need:

- systems designed to provide security and privacy;
- with guarantees that they do;
- used in practice.
The (first) difficulty

Protocols

SSH
TLS
GPG
...
The (first) difficulty

Primitives

\[ X^1 \]

RSA
Elliptic curves
...

Protocols

SSH
TLS
GPG
...

If any link of the chain is broken, everything is.
The (first) difficulty

Implementation  Primitives  Protocols

C++  Java  Python

RSA  Elliptic curves

SSH  TLS  GPG
The (first) difficulty

<table>
<thead>
<tr>
<th>OS</th>
<th>Implementation</th>
<th>Primitives</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>macOS</td>
<td>C++</td>
<td>RSA</td>
<td>SSH</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td>Elliptic curves</td>
<td>TLS</td>
</tr>
<tr>
<td></td>
<td>Python</td>
<td></td>
<td>GPG</td>
</tr>
</tbody>
</table>

If any link of the chain is broken, everything is.
The (first) difficulty

<table>
<thead>
<tr>
<th>Hardware</th>
<th>OS</th>
<th>Implementation</th>
<th>Primitives</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Laptop]</td>
<td>![Linux]</td>
<td>![C++]</td>
<td>![RSA]</td>
<td>![SSH]</td>
</tr>
<tr>
<td>![Server]</td>
<td>![Apple]</td>
<td>![Java]</td>
<td>![Elliptic curves]</td>
<td>![TLS]</td>
</tr>
<tr>
<td>![Mobile]</td>
<td>![Windows]</td>
<td>![Python]</td>
<td></td>
<td>![GPG]</td>
</tr>
</tbody>
</table>
The (first) difficulty

Hardware

OS

Implementation

Primitives

Protocols

Users

- Hardware
- OS
- Implementation
- Primitives
- Protocols
- Users

- Laptop
- Server
- Mobile
- Apple
- Linux
- Windows
- C++
- Java
- Python
- RSA
- Elliptic curves
- SSH
- TLS
- GPG

If any link of the chain is broken, everything is.
The (first) difficulty

If any link of the chain is broken, everything is.
The (first) difficulty

Hardware

OS

Implementation

Primitives

Protocols

Users

Hardware

OS

Implementation

Primitives

Protocols

Users

If any link of the chain is broken, everything is.
The goal

Since the 80’s

Provide guarantees on the protocol assuming that the other layers are secure.
### The goal

**Since the 80’s**

Provide *formal* guarantees on the protocol assuming that the other layers are secure.
The goal

Since the 80’s

Provide formal guarantees on the protocol assuming that the other layers are secure.

$\iff$ a mathematical proof on an abstract model [Goldwasser, Micali, Dolev, Yao]
The goal

Since the 80’s
Provide formal guarantees on the protocol assuming that the other layers are secure. → a mathematical proof on an abstract model [Goldwasser, Micali, Dolev, Yao]

∀A. P || A ⊨ φ
The goal

Since the 80’s

Provide formal guarantees on the protocol assuming that the other layers are secure.

\[ \forall A. \ P \parallel A \models \phi \]

\( P \) - model of the protocol.
The goal

Since the 80’s

Provisional guarantees on the protocol assuming that the other layers are secure.

\[\forall A. \ P \parallel A \models \phi\]

\(P\) - model of the protocol.

\(A\) - attacker model

\[\phi\] - security property

\[\rightarrow\] a mathematical proof on an abstract model [Goldwasser, Micali, Dolev, Yao]
The goal

Since the 80’s

Provide formal guarantees on the protocol assuming that the other layers are secure.

→ a mathematical proof on an abstract model [Goldwasser, Micali, Dolev, Yao]

\[ \forall A. \, P \parallel A \models \phi \]

- \( P \) - model of the protocol.
- \( A \) - attacker model
- \( \phi \) - security property
Second Difficulty - The modeling

Attacker Model
Second Difficulty - The modeling

Attacker Model

Computation Model

- Turing Machines or inference rules
- Assumptions on primitives (RSA)
- Timing attacks
Second Difficulty - The modeling

Attacker Model

Computation Model
- Turing Machines or inference rules
- Assumptions on primitives (RSA)
- Timing attacks

Compromise Model
- Malwares, Keylogger
- Phishing
- Long-term/ephemeral key reveal
<table>
<thead>
<tr>
<th>Computation Model</th>
<th>Compromise Model</th>
<th>Protocol Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing Machines or inference rules</td>
<td>Malwares, Keylogger</td>
<td>Optional behaviours or parameters</td>
</tr>
<tr>
<td>Assumptions on primitives (RSA)</td>
<td>Phishing</td>
<td>Modeling of parsing, serialization</td>
</tr>
<tr>
<td>Timing attacks</td>
<td>Long-term/ephemeral key reveal</td>
<td>Communications channels</td>
</tr>
</tbody>
</table>
Second Difficulty - The modeling

Attacker Model

Computation Model
- Turing Machines or inference rules
- Assumptions on primitives (RSA)
- Timing attacks

Compromise Model
- Malwares, Keylogger
- Phishing
- Long-term/ephemeral key reveal

Security Properties
- Secrecy, PFS, PCS
- Authentication
- Unlinkability

Protocol Model
- Optional behaviours or parameters
- Modeling of parsing, serialization
- Communications channels
The goal

Strong guarantees
Get proofs of security, with all modelings as realistic as possible.
The goal

Strong guarantees
Get proofs of security, with all modelings as realistic as possible.

It is very very very very very very difficult
We want to prove over realistic models that something is impossible, even when considering all possible attackers.

- Undecidable;
- complexity of proofs grows very quickly, and cannot be managed by hand.
Computer-Aided Cryptography (since 2000)
Tools that help us carry-out, verify or automate the proofs.
Computer-Aided Cryptography (since 2000)

Tools that help us carry-out, verify or automate the proofs.

But…

- Inherent trade-off between the realism and automation/proof-size;
Computer-Aided Cryptography (since 2000)
Tools that help us carry-out, verify or automate the proofs.

But...
- Inherent trade-off between the realism and automation/proof-size;
- no single tool will be the best at everything.
The landscape of computer-aided cryptography

Symbolic Tools

Computational Tools

- Proverif, Tamarin, Deepsec...
- EasyCrypt, CryptoVerif, Squirrel...

Attacker

Fixed set of computations

Turing Machines

Compromise

Many

Few

Protocol

Full specification

Core parts in isolation

State of the art

Many tools used successfully, both to prove security or discover new vulnerabilities on complex systems.

Still many limitations, and still very difficult to work on realistic models.
The landscape of computer-aided cryptography

<table>
<thead>
<tr>
<th>Symbolic Tools</th>
<th>Computational Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Proverif, Tamarin, Deepsec, ...)</td>
<td>(EasyCrypt, CryptoVerif, Squirrel ...)</td>
</tr>
</tbody>
</table>

Attacker
Fixed set of computations
Turing Machines
Compromise
Many
Few
Protocol
Full specification
Core parts in isolation

State of the art
Many tools used successfully, both to prove security or discover new vulnerabilities on complex systems.

Still many limitations, and still very difficult to work on realistic models.
# The landscape of computer-aided cryptography

<table>
<thead>
<tr>
<th>Symbolic Tools</th>
<th>Computational Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Proverif, Tamarin, Deepsec, ...)</td>
<td>(EasyCrypt, CryptoVerif, Squirrel ...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Fixed set of computations</th>
<th>Turing Machines</th>
</tr>
</thead>
</table>
The landscape of computer-aided cryptography

Symbolic Tools
(Proverif, Tamarin, Deepsec, ...)

Computational Tools
(EasyCrypt, CryptoVerif, Squirrel ...)

Attacker
Fixed set of computations
Compromise
Many

Turing Machines
Few
The landscape of computer-aided cryptography

<table>
<thead>
<tr>
<th>Symbolic Tools</th>
<th>Computational Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Proverif, Tamarin, Deepsec, ...)</td>
<td>(EasyCrypt, CryptoVerif, Squirrel ...)</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Attacker</td>
<td>Fixed set of computations</td>
</tr>
<tr>
<td>Compromise</td>
<td>Many</td>
</tr>
<tr>
<td>Protocol</td>
<td>Full specification</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Turing Machines</td>
</tr>
<tr>
<td></td>
<td>Few</td>
</tr>
<tr>
<td></td>
<td>Core parts in isolation</td>
</tr>
</tbody>
</table>
# The landscape of computer-aided cryptography

## Symbolic Tools
(Proverif, Tamarin, Deepsec, ...)

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Fixed set of computations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compromise</td>
<td>Many</td>
</tr>
<tr>
<td>Protocol</td>
<td>Full specification</td>
</tr>
</tbody>
</table>

## Computational Tools
(EasyCrypt, CryptoVerif, Squirrel ...)

<table>
<thead>
<tr>
<th>Turing Machines</th>
<th>Few</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core parts in isolation</td>
<td></td>
</tr>
</tbody>
</table>

## State of the art

- Many tools used **successfully**, both to prove security or discover new vulnerabilities on complex systems.

---

---
The landscape of computer-aided cryptography

Symbolic Tools
(Proverif, Tamarin, Deepsec, ...)

Computational Tools
(EasyCrypt, CryptoVerif, Squirrel ...)

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Fixed set of computations</th>
<th>Turing Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compromise</td>
<td>Many</td>
<td>Few</td>
</tr>
<tr>
<td>Protocol</td>
<td>Full specification</td>
<td>Core parts in isolation</td>
</tr>
</tbody>
</table>

State of the art

- Many tools used **successfully**, both to prove security or discover new vulnerabilities on complex systems.
- Still many **limitations**, and still very difficult to work on realistic models.
What I have been doing

<table>
<thead>
<tr>
<th>Theory - Make proofs easier for realistic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition results to cut computational and symbolic proofs into modular pieces; [Comon, J., Scerri - CCS’20]</td>
</tr>
<tr>
<td>Automation of basic proofs steps in the computational world; [Barthe, J., Kremer - LICS’20, TOCL] &amp; [BGJKS - CSF’19] &amp; [BFGGJS - CCS’18]</td>
</tr>
<tr>
<td>A new computational tool allowing for easier proofs of complex protocols; Squirrel Prover [Baelde, Delaune, J., Koutsos, Moreau - S&amp;P’21]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Practice - Actually make proofs for realistic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensive analysis in Proverif of multi-factor authentication; 6000 scenarios generated and verified in 5 minutes [Kremer, J. - CSF’18, TOPS]</td>
</tr>
<tr>
<td>Modular analysis of SSH in Squirrel, with optional feature of agent forwarding; carried out first in the composition paper and then in the Squirrel one.</td>
</tr>
</tbody>
</table>
What I have been doing

Theory - Make proofs easier for realistic models

- Composition results to cut computational and symbolic proofs into modular pieces;
  [Comon, J., Scerri - CCS’20]

Practice - Actually make proofs for realistic models

- Extensive analysis in Proverif of multi-factor authentication;
  [Kremer, J. - CSF’18, TOPS]
- Modular analysis of SSH in Squirrel, with optional feature of agent forwarding;
  Carried out first in the composition paper and then in the Squirrel one.
What I have been doing

<table>
<thead>
<tr>
<th>Theory - Make proofs easier for realistic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Composition results to cut computational and symbolic proofs into <em>modular</em> pieces;</td>
</tr>
<tr>
<td>[Comon, J., Scerri - CCS'20]</td>
</tr>
<tr>
<td>- <strong>Automation</strong> of basic proofs steps in the computational world;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Practice - Actually make proofs for realistic models</th>
</tr>
</thead>
</table>
What I have been doing

Theory - Make proofs easier for realistic models

- Composition results to cut computational and symbolic proofs into modular pieces;
  [Comon, J., Scerri - CCS'20]
- Automation of basic proofs steps in the computational world;
- A new computational tool allowing for easier proofs of complex protocols;
  Squirrel Prover [Baelde, Delaune, J., Koutsos, Moreau - S&P’21]

Practice - Actually make proofs for realistic models
What I have been doing

<table>
<thead>
<tr>
<th>Theory - Make proofs easier for realistic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Composition results to cut computational and symbolic proofs into <strong>modular</strong> pieces;</td>
</tr>
<tr>
<td>[Comon, J., Scerri - CCS’20]</td>
</tr>
<tr>
<td>■ <strong>Automation</strong> of basic proofs steps in the computational world;</td>
</tr>
<tr>
<td>■ A new computational tool allowing for easier proofs of complex protocols;</td>
</tr>
<tr>
<td><strong>Squirrel Prover</strong> [Baelde, Delaune, J., Koutsos, Moreau - S&amp;P’21]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Practice - Actually make proofs for realistic models</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ <strong>Extensive</strong> analysis in Proverif of multi-factor authentication;</td>
</tr>
<tr>
<td>6000 scenarios generated and verified in 5 minutes [Kremer, J. - CSF’18, TOPS]</td>
</tr>
</tbody>
</table>
### What I have been doing

#### Theory - Make proofs easier for realistic models

- Composition results to cut computational and symbolic proofs into **modular** pieces;
  
  [Comon, J., Scerri - CCS’20]

- **Automation** of basic proofs steps in the computational world;
  

- A new computational tool allowing for easier proofs of complex protocols;
  
  **Squirrel Prover** [Baelde, Delaune, J., Koutsos, Moreau - S&P’21]

#### Practice - Actually make proofs for realistic models

- **Extensive** analysis in Proverif of multi-factor authentication;
  
  6000 scenarios generated and verified in 5 minutes [Kremer, J. - CSF’18, TOPS]

- Modular analysis of **SSH** in Squirrel, with optional feature of agent forwarding;
  
  Carried out first in the composition paper and then in the Squirrel one.
A new attacker model

The tools should be able to provide guarantees against quantum attackers.
A new attacker model

The tools should be able to provide guarantees against quantum attackers.

- What changes with a quantum attackers?
Today’s presentation

A new attacker model

The tools should be able to provide guarantees against quantum attackers.

- What changes with a quantum attackers?
- Can tools already provide guarantees about them?
A new attacker model

The tools should be able to provide guarantees against quantum attackers.

- What changes with a quantum attacker?
- Can tools already provide guarantees about them?
- If not, what can we do to fix them?
What is the fuss about quantum attackers?
<table>
<thead>
<tr>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No scaling quantum computers</td>
</tr>
</tbody>
</table>
Quantum computers

When?
No scaling quantum computers yet...
### Quantum computers

**When?**

No scaling quantum computers *yet...*

**The issue**

Quantum computers allow for a **significant speed up** for solving many problems

⇒ breaks RSA, computes discrete logarithms...
Quantum computers

When?
No scaling quantum computers yet...

The issue
Quantum computers allow for a significant speed up for solving many problems
  ⇒ breaks RSA, computes discrete logarithms...

→ We need new primitives, new protocols and new proofs.
Attacker models

Symbolic Tools
(Proverif, Tamarin, Deepsec, ...)

Attacker
Fixed set of possible computations on abstract messages

Computational Tools
(EasyCrypt, CryptoVerif, ...)

Turing Machines on bitstrings

Post-quantum?
Abstract reasoning still valid
<table>
<thead>
<tr>
<th>Attacker models</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Symbolic Tools</strong></th>
<th><strong>Computational Tools</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Proverif, Tamarin, Deepsec, ... )</td>
<td>(EasyCrypt, CryptoVerif, ... )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Fixed set of possible computations on abstract messages</th>
<th>Turing Machines on bitstrings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-quantum?</td>
<td>Abstract reasoning still valid</td>
<td></td>
</tr>
<tr>
<td>Attacker models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Symbolic Tools</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Proverif, Tamarin, Deepsec, ...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attacker</td>
<td>Fixed set of possible computations on abstract messages</td>
<td></td>
</tr>
<tr>
<td>Post-quantum?</td>
<td>Abstract reasoning still valid</td>
<td></td>
</tr>
<tr>
<td><strong>Computational Tools</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(EasyCrypt, CryptoVerif, ...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turing Machines on bitstrings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum Turing Machines</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A first look at classical computational proofs
A first look at classical computational proofs

Computational Hardness Assumption $\rightarrow$ Protocol Security

Attacker on Protocol
A first look at classical computational proofs

Computational Hardness Assumption → Protocol Security

Attacker on Assumption → reduction → Attacker on Protocol
### Two ingredients

- **An assumption**
  
  (a computational assumption that holds for any attacker, e.g. RSA is unbreakable)
Classical Proofs

Two ingredients

- An assumption
  (a computational assumption that holds for any attacker, e.g. RSA is unbreakable)

- A reduction
  (the construction of a new attacker using the one against the assumption,
  similar to NP-hardness proofs or undecidability proofs)
Classical Proofs

Two ingredients

- An assumption over quantum computers
  (a computational assumption that holds for any attacker, e.g. RSA is unbreakable)

- A reduction over quantum computers
  (the construction of a new attacker using the one against the assumption,
  similar to NP-hardness proofs or undecidability proofs)
### Two ingredients

- **An assumption** *over quantum computers*
  
  (a computational assumption that holds for any attacker, e.g. RSA is unbreakable)

- **A reduction** *over quantum computers*
  
  (the construction of a new attacker using the one against the assumption, similar to NP-hardness proofs or undecidability proofs)

### A tale of two issues

- No drop in quantum replacement for some classical assumptions (DDH).
## Classical Proofs

### Two ingredients

- **An assumption over quantum computers**
  
  (a computational assumption that holds for any attacker, e.g. RSA is unbreakable)

- **A reduction over quantum computers**
  
  (the construction of a new attacker using the one against the assumption, similar to NP-hardness proofs or undecidability proofs)

### A tale of two issues

- No drop in quantum replacement for some classical assumptions (DDH).

- There are ways to manipulate a classical attacker that cannot be done with a quantum one.
What is a classical attacker?

Probabilistic attacker model

A deterministic computer \( \mathcal{A} \) with a random string \( \rho \) and inputs \( \vec{i} \)

\[
\mathcal{A}(\rho, \vec{i})
\]
What is a classical attacker?

### Probabilistic attacker model

A deterministic computer $\mathcal{A}$ with a random string $\rho$ and inputs $\vec{i}$

$\mathcal{A}(\rho, \vec{i})$

### Allows to simulate weird executions

Run twice the attacker with the same source of randomness on two distinct inputs: $\mathcal{A}(\rho, \vec{i}_1)$ and $\mathcal{A}(\rho, \vec{i}_2)$
What is a classical attacker?

**Probabilistic attacker model**

A deterministic computer $\mathcal{A}$ with a random string $\rho$ and inputs $\vec{i}$

$$\mathcal{A}(\rho, \vec{i})$$

**Allows to simulate weird executions**

Run twice the attacker with the **same source** of randomness on **two distinct** inputs:

$\mathcal{A}(\rho, \vec{i}_1)$ and $\mathcal{A}(\rho, \vec{i}_2)$

Impossible computation with a quantum computer
What is a quantum attacker?

It is impossible to

- Run twice a quantum computer with fixed randomness
- Duplicate a quantum state (no-cloning theorem)

Reductions must not use techniques relying on this (e.g., rewinding)
It is impossible to

- Run twice a quantum computer with fixed randomness
What is a quantum attacker?

It is impossible to

- Run twice a quantum computer with fixed randomness
- Duplicate a quantum state (no-cloning theorem)
What is a quantum attacker?

It is impossible to

- Run twice a quantum computer with fixed randomness
- Duplicate a quantum state (no-cloning theorem)

Reductions must not use techniques relying on this (e.g., rewinding)
Many pitfalls

Must be careful about

- manipulations of the attacker’s state;
- mentions of the attacker’s randomness;
Many pitfalls

Must be careful about

- manipulations of the attacker’s state;
- mentions of the attacker’s randomness;
- arguments about numbers of queries made to an oracle (QROM);
- arguments about complexity classes.
Quantum-sound reductions

Many pitfalls

Must be careful about

- manipulations of the attacker’s state;
- mentions of the attacker’s randomness;
- arguments about numbers of queries made to an oracle (QROM);
- arguments about complexity classes.

→ The computational tools do this kind of things...
Contributions

Our contributions\(^1\)

\(^1\)Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
Contributions

Our contributions\(^1\)

- Take the **BC logic** - a logic for deriving computational security guarantees

\(^1\)Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
Our contributions¹

- Take the BC logic - a logic for deriving computational security guarantees
- Make it sound for quantum attackers

¹Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
Contributions

Our contributions\(^1\)

- Take the **BC logic** - a logic for deriving computational security guarantees
- Make it sound for quantum attackers
- Take the **Squirrel Prover** - an interactive prover for the BC logic

\(^1\)Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
Contributions

Our contributions\(^1\)

- Take the *BC logic* - a logic for deriving computational security guarantees
- Make it sound for quantum attackers
- Take the *Squirrel Prover* - an interactive prover for the BC logic
- Extend it to support the adapted PQ sound logic

\(^1\)Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
## Contributions

<table>
<thead>
<tr>
<th>Our contributions¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Take the BC logic - a logic for deriving computational security guarantees</td>
</tr>
<tr>
<td>- Make it sound for quantum attackers</td>
</tr>
<tr>
<td>- Take the Squirrel Prover - an interactive prover for the BC logic</td>
</tr>
<tr>
<td>- Extend it to support the adapted PQ sound logic</td>
</tr>
<tr>
<td>- Use it on some new protocols - KEM based post-quantum key exchanges</td>
</tr>
</tbody>
</table>

¹Joint work with Cas Cremers, Caroline Fontaine, and discussions with Hubert Comon.
Some related work

  - Identified the no cloning theorem as an issue.
Some related work

- John Watrous. Zero-knowledge against quantum attacks. → Identified the no cloning theorem as an issue.


EasyPQC - [BBFGHKSWZ - CCS'21] (parallel work) → Post-quantum sound EasyCrypt - hard to scale to protocols
Some related work

  → Identified the no cloning theorem as an issue.


  → Identified classes of valid reductions for pen and paper proofs.

- EasyPQC - [BBFGHKSWZ - CCS’21] (parallel work)
  → Post-quantum sound EasyCrypt - hard to scale to protocols
A post-quantum BC logic
The BC logic

The BC logic\(^2\)

A **first-order logic** to prove the security of protocols.

\(^2\)[Bana, Comon-CCS'14]
The BC logic

The BC logic\(^2\)

A first-order logic to prove the security of protocols.

\[\rightarrow\text{ a proof implies the existence of a reduction.}\]

\(^2\)[Bana, Comon-CCS’14]
The BC logic

A first-order logic to prove the security of protocols.

→ a proof implies the existence of a reduction.

A computationally sound logic

Three main ingredients:

- terms, and their interpretation so that terms can syntactically describe all behaviours of a protocol;
  → if there exists an attack on the protocol, we can see it on the terms.

\[2\text{[Bana, Comon-CCS’14]}\]
The BC logic

The BC logic\(^2\)

A **first-order logic** to prove the security of protocols.

\[ \iff \text{a proof implies the existence of a reduction.} \]

A computationally sound logic

Three main ingredients:

- terms, and their **interpretation** so that terms can syntactically describe all behaviours of a protocol;
  \[ \iff \text{if there exists an attack on the protocol, we can see it on the terms.} \]

- logical **predicates and rules** (with axioms about e.g. RSA) to reason over the terms;

\(^2\)\cite{Bana:Comon:CCS’14}
The BC logic

A first-order logic to prove the security of protocols.

→ a proof implies the existence of a reduction.

A computationally sound logic

Three main ingredients:

- terms, and their interpretation so that terms can syntactically describe all behaviours of a protocol;
  
  → if there exists an attack on the protocol, we can see it on the terms.

- logical predicates and rules (with axioms about e.g. RSA) to reason over the terms;

- prove the soundness of the rules, i.e., they correspond to valid reduction.
  
  → if there is an attack on the protocol, there is an attack against the axioms.

\(^2\) [Bana, Comon-CCS’14]
Make it post-quantum sound

- New primitives;
  - design new axioms and rules.
Going post-quantum

Make it post-quantum sound

- New primitives;
  - design new axioms and rules.

- Verify the post-quantum soundness of the rules;
  - do we manipulate the attacker in a bad way?
Going post-quantum

Make it post-quantum sound

- New primitives;
  - design new axioms and rules.
- Verify the post-quantum soundness of the rules;
  - do we manipulate the attacker in a bad way?
- Verify the term interpretation;
  - do we manipulate the attacker in a bad way?
Going post-quantum

Make it post-quantum sound

- New primitives;
  - design new axioms and rules. straightforward
- Verify the post-quantum soundness of the rules;
  - do we manipulate the attacker in a bad way?
- Verify the term interpretation;
  - do we manipulate the attacker in a bad way?
Make it post-quantum sound

- New primitives;
  - design new axioms and rules. *straightforward*

- Verify the post-quantum soundness of the rules;
  - do we manipulate the attacker in a bad way? *No*

- Verify the term interpretation;
  - do we manipulate the attacker in a bad way?
Going post-quantum

Make it post-quantum sound

- New primitives; design new axioms and rules. straightforward
- Verify the post-quantum soundness of the rules; do we manipulate the attacker in a bad way? No
- Verify the term interpretation; do we manipulate the attacker in a bad way? Yes
The BC logic

The BC logic\(^3\)

\(^3\)[Bana, Comon-CCS'14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

\[\text{[Bana, Comon-CCS'14]}\]
The BC logic

<table>
<thead>
<tr>
<th>Classical proofs</th>
<th>BC terms</th>
</tr>
</thead>
</table>

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

[^3]: [Bana, Comon-CCS'14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

Classical proofs    BC terms

\[ sk \gets \{0, 1\}^n \]

\(^3[\text{Bana, Comon-CCS'14}]\)
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

Classical proofs

\[ sk \leftarrow \{0,1\}^n \]

BC terms

\[ sk \]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

<table>
<thead>
<tr>
<th>Classical proofs</th>
<th>BC terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>( sk \leftarrow {0, 1}^\eta )</td>
<td>( sk )</td>
</tr>
<tr>
<td>( m \leftarrow A(1^\eta) )</td>
<td></td>
</tr>
</tbody>
</table>

\(^3\)[Bana, Comon-CCS'14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

<table>
<thead>
<tr>
<th>Classical proofs</th>
<th>BC terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>( sk \xleftarrow{$} {0, 1}^\eta )</td>
<td>( sk )</td>
</tr>
<tr>
<td>( m \xleftarrow{$} A(1^\eta) )</td>
<td>( \text{att}_0() )</td>
</tr>
</tbody>
</table>

\[ [\text{Bana, Comon-CCS'14}] \]
Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

<table>
<thead>
<tr>
<th>Classical proofs</th>
<th>BC terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk \leftarrow {0, 1}^\eta$</td>
<td>sk</td>
</tr>
<tr>
<td>$m \leftarrow A(1^\eta)$</td>
<td>att$_0$( )</td>
</tr>
<tr>
<td>$t \leftarrow \text{enc}(m, sk)$</td>
<td></td>
</tr>
</tbody>
</table>

$^3$[Bana, Comon-CCS’14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

<table>
<thead>
<tr>
<th>Classical proofs</th>
<th>BC terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk \leftarrow {0, 1}^\eta$</td>
<td>$sk$</td>
</tr>
<tr>
<td>$m \leftarrow A(1^n)$</td>
<td>$att_0()$</td>
</tr>
<tr>
<td>$t \leftarrow \text{enc}(m, sk)$</td>
<td>$\text{enc}(att_0(), r, sk)$</td>
</tr>
</tbody>
</table>

$^3$[Bana, Comon-CCS’14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

Classical proofs

<table>
<thead>
<tr>
<th>SK</th>
<th>${0, 1}^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>$A(1^n)$</td>
</tr>
<tr>
<td>t</td>
<td>$\text{enc}(m, sk)$</td>
</tr>
<tr>
<td>x</td>
<td>$A(t)$</td>
</tr>
</tbody>
</table>

BC terms

<table>
<thead>
<tr>
<th>sk</th>
<th>$\text{att}_0()$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{enc}(\text{att}_0(), r, sk)$</td>
<td></td>
</tr>
</tbody>
</table>

$^3$[Bana, Comon-CCS'14]
The BC logic

Protocols are now expressed only with terms, i.e., purely syntactic construct, where everything becomes pure functional calls.

### Classical proofs

- $sk \leftarrow \{0, 1\}$
- $m \leftarrow A(1^n)$
- $t \leftarrow enc(m, sk)$
- $x \leftarrow A(t)$

### BC terms

- $sk$
- $att_0()$
- $enc(att_0(), r, sk)$
- $att_1(enc(att_0(), r, sk))$

---

$^3$[Bana, Comon-CCS'14]
A protocol

```
new sk;
in(x);
if x = sk then
  out(ko)
else
  out(ok)
```
A protocol

new sk;
in(x);
if $x = sk$ then
  out(ko)
else
  out(ok)

Becomes a term

if $(\text{att}_0()) = sk$ then $ko$ else $ok$
Some rules

$\text{Refl}\;u \sim \text{ind}(t) \sim \false$ when $n$ does not occur in $t$

$\text{If-ff}\;\phi \sim \false \quad u \sim v$ if $\phi$ then $u$ else $v$ \sim w

$\text{Refl}\;\text{ok} \sim \text{ok}$

$\text{if att}_0() = \text{sk} \text{ then } \text{ko} \text{ else } \text{ok} \sim \text{ok}$
Some rules

\[
\text{Refl} \\
\hline
u \sim u
\]
Some rules

- **Refl**

  \[
  \frac{}{u \sim u}
  \]

- **=ind**

  \[
  \frac{}{(t \div n) \sim \text{false} \quad \text{when } n \text{ does not occur in } t}
  \]
Some rules

Refl

\[ u \sim u \]

\[ (t \doteq n) \sim \text{false} \]

when \( n \) does not occur in \( t \)

\[ \phi \sim \text{false} \quad u \sim w \]

If-f

if \( \phi \) then \( u \) else \( v \sim w \)
Some rules

Refl

\[ u \sim u \]

\( \text{=ind} \)

\[ (t \div n) \sim \text{false} \]

\( \text{when } n \text{ does not occur in } t \)

\( \text{If-f} \)

\[ \phi \sim \text{false} \quad u \sim w \]

\( \text{if } \phi \text{ then } u \text{ else } v \sim w \)

if \( \text{att}_0() = \text{sk} \) then \( \text{ko} \) else \( \text{ok} \sim \text{ok} \)
Some rules

**Refl**

\[ u \sim u \]

**=ind**

\[ (t \div n) \sim \text{false} \]

when \( n \) does not occur in \( t \)

**If-f**

\[ \phi \sim \text{false} \quad u \sim w \]

if \( \phi \) then \( u \) else \( v \sim w \)

**If-f**

\[ \text{if att}_0() = \text{sk then } ko \text{ else } ok \sim ok \]
Some rules

**Refl**

\[
\begin{align*}
  u & \sim u \\
  (t \not\equiv n) & \sim \text{false} \\
\end{align*}
\]

**=ind**

when \( n \) does not occur in \( t \)

**If-f**

\[
\begin{align*}
  \phi & \sim \text{false} \\
  u & \sim w \\
\end{align*}
\]

if \( \phi \) then \( u \) else \( v \) \sim w

**If-f**

\[
\begin{align*}
  (\text{att}_0() = \text{sk}) & \sim \text{false} \\
\end{align*}
\]

if \( \text{att}_0() = \text{sk} \) then \( \text{ko} \) else \( \text{ok} \) \sim \text{ok}
Some rules

Refl

\[ u \sim u \]

\[ (t \div n) \sim \text{false when } n \text{ does not occur in } t \]

=ind

\[ \phi \sim \text{false } \quad u \sim w \]

If-f

\[ \text{if } \phi \text{ then } u \text{ else } v \sim w \]

If-f

\[ (\text{att}_0() = \text{sk}) \sim \text{false } \quad \text{ok} \sim \text{ok} \]

\[ \text{if } \text{att}_0() = \text{sk} \text{ then } \text{ko} \text{ else } \text{ok} \sim \text{ok} \]
Some rules

**Refl**

\[ u \sim u \]

**=ind**

\[ (t \neq n) \sim \text{false} \]

when \( n \) does not occur in \( t \)

**If-f**

\[ \phi \sim \text{false} \quad u \sim w \]

if \( \phi \) then \( u \) else \( v \sim w \)

**If-f**

\[ \text{att}_0() = \text{sk} \sim \text{false} \]

\[ \text{ok} \sim \text{ok} \]

if \( \text{att}_0() = \text{sk} \) then \( \text{ko} \) else \( \text{ok} \sim \text{ok} \)
Some rules

Refl
$u \sim u$

=ind
$(t \neq n) \sim false$

when $n$ does not occur in $t$

If-f
$\phi \sim false$

$u \sim w$

if $\phi$ then $u$ else $v \sim w$

Refl
$ok \sim ok$

If-f
$(att_0() = sk) \sim false$

if $att_0() = sk$ then $ko$ else $ok \sim ok$
Does this allow to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[
\text{enc}(\text{att}_1(\text{enc}(\text{att}_0(), r), \text{sk}), r', \text{sk})
\]

The logic quantifies over all sets of potential values of \(\text{att}_1\) and \(\text{att}_0\) ↦ → all possible Turing machines \(T_{\text{att}_0}\) and \(T_{\text{att}_1}\), and thus all attackers. But...

In the real world, we have a stateful interactive probabilistic attacker \(A\). In the BC world, we have two stateless (because a call to \(\text{att}_i\) must be pure) and independent deterministic attackers \(T_{\text{att}_i}\) that share a source of randomness. Solved by specifying that \(T_{\text{att}_1}\) always starts by recomputing the state of \(T_{\text{att}_0}\).
But wait...

Does this allows to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[ \text{att}_0() \]
But wait...

Does this allows to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[ \text{enc}((\text{att}_0(), r, sk), r', sk) \]
But wait...

Does this allow to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[ \text{att}_1(\text{enc}(\text{att}_0(), r, \text{sk})) \]
But wait...

Does this allows to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[ \text{enc}(\text{att}_1(\text{enc}(\text{att}_0(), r, sk)), r', sk) \]
But wait... Does this allow to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

$$\text{enc}(\text{enc}(\text{att}_0(), r, sk), r', sk)$$

The logic quantifies over all sets of potential values of $\text{att}_1$ and $\text{att}_0$

$\leftrightarrow$ all possible Turing machines $T_{\text{att}_0}$ and $T_{\text{att}_1}$, and thus all attackers.
**But wait...**

Does this allows to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[
\text{enc}(\text{att}_1(\text{enc}(\text{att}_0(), r, sk)), r', sk)
\]

The logic quantifies over all sets of potential values of \( \text{att}_1 \) and \( \text{att}_0 \)

\[\mapsto\text{all possible Turing machines } \mathcal{T}_{\text{att}_0} \text{ and } \mathcal{T}_{\text{att}_1}, \text{ and thus all attackers.}\]

**But...**

- In the real world, we have a **stateful interactive probabilistic attacker** \( \mathcal{A} \).
But wait... Does this allow to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

$$\text{enc}(\text{att}_1(\text{enc}(\text{att}_0(), r, sk)), r', sk)$$

The logic quantifies over all sets of potential values of att$_1$ and att$_0$

$$\rightarrow$$ all possible Turing machines $T_{\text{att}_0}$ and $T_{\text{att}_1}$, and thus all attackers.

But...

- In the real world, we have a **stateful interactive** probabilistic attacker $A$.
- In the BC world, we have **two stateless** (because a call to att$_i$ must be pure) and independent deterministic attackers $T_{\text{att}_i}$ that share a source of randomness.
But wait... Does this allows to capture real life behaviours?

A protocol where we encrypt two consecutive attacker chosen values:

\[ \text{enc}(\text{att}_1(\text{enc}(\text{att}_0(), r, sk)), r', sk) \]

The logic quantifies over all sets of potential values of att_1 and att_0

\[ \mapsto \text{all possible Turing machines } T_{\text{att}_0} \text{ and } T_{\text{att}_1}, \text{ and thus all attackers.} \]

But...

- In the real world, we have a \textit{stateful interactive} probabilistic attacker A.
- In the BC world, we have \textit{two stateless} (because a call to att_i must be pure) and independent deterministic attackers $T_{\text{att}_i}$ that share a source of randomness.

\[ \mapsto \text{Solved by specifying that } T_{\text{att}_1} \text{ always starts by recomputing the state of } T_{\text{att}_0} \]
Real world interaction vs BC interaction

We compute twice $A_0(\rho_r, \eta)$ to reconstruct its state.

Impossible with a quantum attacker.
We compute twice $A_0(\rho_r, \eta)$ to reconstruct its state. Impossible with a quantum attacker.
Real world interaction vs BC interaction

We compute twice $A_0(\rho_r, \eta)$ to reconstruct its state.
Real world interaction vs BC interaction

We compute twice $A_0(\rho_r, \eta)$ to reconstruct its state.

Impossible with a quantum attacker
## First issue

Behind the curtain, the interpretation of terms crucially rely on two facts:

- we can see a probabilistic attacker as some deterministic $\mathcal{A}(1^n, \rho_r)$,
- and run it multiple times with the same randomness to reconstruct internal states.
## First issue

Behind the curtain, the interpretation of terms crucially rely on two facts:

- we can see a probabilistic attacker as some deterministic $\mathcal{A}(1^n, \rho_r)$,
- and run it multiple times with the same randomness to reconstruct internal states.

The two **impossible** operations with a quantum attacker!
First issue

Behind the curtain, the interpretation of terms crucially rely on two facts:

- we can see a probabilistic attacker as some deterministic $\mathcal{A}(1^n, \rho_r)$,
- and run it multiple times with the same randomness to reconstruct internal states.

The two impossible operations with a quantum attacker!

Our main contribution

An interpretation sound for interactive black-box attackers, where the interpretation directly depends a single interactive Turing Machine $\mathcal{T}_A$, instead of many $\mathcal{T}_{\text{att}_i}$. 
New (natural) interpretation

But the old rules break down...

\[
\text{if } \text{att}^0(\text{sk}) = \text{att}^1() \text{ then } \text{ko} \text{ else } \text{ok}
\]
New (natural) interpretation

But the old rules break down...
New (natural) interpretation

\[
\text{But the old rules break down...}
\]

\[
\text{att}_0(sk), \text{ if } \text{att}_1() = sk \text{ then ko else ok}
\]
New (natural) interpretation

But the old rules break down...

\[ \text{att}_0(\text{sk}), \text{if att}_1() = \text{sk then ko else ok} \]

\[ \sim \]

\[ \text{att}_0(\text{sk}), \text{ok} \]
A cascade of changes
A cascade of changes

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
A cascade of changes

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
  - we must forbid such things, that model a rewinding
If it could have been that simple...

A cascade of changes

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
  - \(\Rightarrow\) we must forbid such things, that model a rewinding
- What is the meaning of the sequence \((\text{att}_0(\text{ok}), \text{att}_1())\)?

He single interactive attacker will know how many times it was called on both sides!
If it could have been that simple...

**A cascade of changes**

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
  - → we must forbid such things, that model a rewinding

- What is the meaning of the sequence \((\text{att}_0(\text{ok}), \text{att}_1())\)?
  - → \text{att}_1 should depend on \text{ok}, as the machine that will interpret it will have seen it in the first step.
A cascade of changes

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
  \(\iff\) we must forbid such things, that model a rewinding

- What is the meaning of the sequence \((\text{att}_0(\text{ok}), \text{att}_1())\)?
  \(\iff\) \text{att}_1 should depend on \text{ok}, as the machine that will interpret it will have seen it in the first step.

- What is the validity of the formula \((\text{att}_0() \div n) \sim (\text{att}_1(\text{att}_0()) \div n)\)?
If it could have been that simple...

A cascade of changes

- What is the meaning of the sequence \((\text{att}_1(\text{ok}), \text{att}_1(\text{ko}))\)?
  - We must forbid such things, that model a rewinding

- What is the meaning of the sequence \((\text{att}_0(\text{ok}), \text{att}_1())\)?
  - \text{att}_1 should depend on \text{ok}, as the machine that will interpret it will have seen it in the first step.

- What is the validity of the formula \((\text{att}_0() \div n) \sim (\text{att}_1(\text{att}_0()) \div n)\)?
  - He single interactive attacker will know how many time it was called on both sides!
Syntactic conditions

A set of three simple syntactic conditions over terms and formulas.

- Consistency: all occurrences of an attacker with the same arguments:
  \[\forall \vec{t}, \text{att}_i(x) \in \vec{t} \land \text{att}_i(y) \in \vec{t} \Rightarrow x = y\]

- Monotonicity: inputs of an attacker are a prefix of the inputs of another attacker:
  \[\forall \vec{t}, i < j, \text{att}_i(u_1, \ldots, u_i) \in \vec{t} \land \text{att}_j(u'_1, \ldots, u'_j) \in \vec{t} \Rightarrow u_1 = u'_1 \land \cdots \land u_i = u'_i\]

- Balance: same number of calls to the attacker on both sides of every attack:
  \[\forall i, \text{att}_i \in \vec{u} \iff \text{att}_i \in \vec{v}\]
Our solution

<table>
<thead>
<tr>
<th>Syntactic conditions</th>
<th>A set of three simple syntactic conditions over terms and formulas.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consistency</strong></td>
<td>- all occurrences of a $\text{att}_i$ for some $i$ occurs with the same arguments;</td>
</tr>
<tr>
<td><strong>Monotonicity</strong></td>
<td>- inputs of $\text{att}_j$ are a prefix of the inputs of $\text{att}_j$, $j &gt; i$;</td>
</tr>
<tr>
<td><strong>Balance</strong></td>
<td>- Same number of calls to the attacker on both sides of every $\vec{u} \sim \vec{v}$.</td>
</tr>
</tbody>
</table>
Our solution

**Syntactic conditions**

A set of three simple syntactic conditions over terms and formulas.

- **Consistency** - all occurrences of a $\text{att}_i$ for some $i$ occurs with the same arguments;

  $$\forall \vec{t}, \text{att}_i(x) \in \vec{t} \land \text{att}_i(y) \in \vec{t} \Rightarrow x = y$$
Our solution

Syntactic conditions
A set of three simple syntactic conditions over terms and formulas.

- **Consistency** - all occurrences of a $\text{att}_i$ for some $i$ occurs with the same arguments;
  \[
  \forall \vec{t}, \text{att}_i(x) \in \vec{t} \land \text{att}_i(y) \in \vec{t} \Rightarrow x = y
  \]

- **Monotonicity** - inputs of $\text{att}_i$ are a prefix of the inputs of $\text{att}_j, j > i$;
Our solution

Syntactic conditions

A set of three simple syntactic conditions over terms and formulas.

- **Consistency** - all occurrences of a \( \text{att}_i \) for some \( i \) occurs with the same arguments;
  \[
  \forall \vec{t}, \text{att}_i(x) \in \vec{t} \land \text{att}_i(y) \in \vec{t} \Rightarrow x = y
  \]

- **Monotonicity** - inputs of \( \text{att}_i \) are a prefix of the inputs of \( \text{att}_j, j > i \);
  \[
  \forall \vec{t}, i < j, \text{att}_i(u_1, \ldots, u_i) \in \vec{t} \land \text{att}_j(u'_1, \ldots, u'_j) \in \vec{t} \Rightarrow u_1 = u'_1 \land \cdots \land u_i = u'_i
  \]
Syntactic conditions

A set of three simple syntactic conditions over terms and formulas.

- **Consistency** - all occurrences of $\text{att}_i$ for some $i$ occurs with the same arguments;
  \[
  \forall \vec{t}, \text{att}_i(x) \in \vec{t} \wedge \text{att}_i(y) \in \vec{t} \Rightarrow x = y
  \]

- **Monotonicity** - inputs of $\text{att}_i$ are a prefix of the inputs of $\text{att}_j$, $j > i$;
  \[
  \forall \vec{t}, i < j, \text{att}_i(u_1, \ldots, u_i) \in \vec{t} \wedge \text{att}_j(u'_1, \ldots, u'_j) \in \vec{t} \Rightarrow u_1 = u'_1 \wedge \cdots \wedge u_i = u'_i
  \]

- **Balance** - Same number of calls to the attacker on both sides of every $\vec{u} \sim \vec{v}$. 
### Syntactic conditions

A set of three simple syntactic conditions over terms and formulas.

- **Consistency** - all occurrences of a $\text{att}_i$ for some $i$ occurs with the same arguments;
  \[
  \forall \vec{t}, \text{att}_i(x) \in \vec{t} \land \text{att}_i(y) \in \vec{t} \Rightarrow x = y
  \]

- **Monotonicity** - inputs of $\text{att}_i$ are a prefix of the inputs of $\text{att}_j$, $j > i$;
  \[
  \forall \vec{t}, i < j, \text{att}_i(u_1, \ldots, u_i) \in \vec{t} \land \text{att}_j(u'_1, \ldots, u'_j) \in \vec{t} \Rightarrow u_1 = u'_1 \land \cdots \land u_i = u'_i
  \]

- **Balance** - Same number of calls to the attacker on both sides of every $\vec{u} \sim \vec{v}$.
  \[
  \forall i, \text{att}_i \in \vec{u} \iff \text{att}_i \in \vec{v}
  \]
Our solution

<table>
<thead>
<tr>
<th>The conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Those specific three conditions were chosen because they are:</td>
</tr>
</tbody>
</table>
Our solution

The conditions

Those specific three conditions were chosen because they are:

- **Necessary**, otherwise one can write terms that don’t have any interpretation in the quantum world;
Our solution

<table>
<thead>
<tr>
<th>The conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Those specific three conditions were chosen because they are:</td>
</tr>
<tr>
<td>■ <strong>Necessary</strong>, otherwise one can write terms that don’t have any interpretation in the quantum world;</td>
</tr>
<tr>
<td>■ <strong>Sufficient</strong> to obtain the soundness of the BC logic;</td>
</tr>
</tbody>
</table>
Our solution

The conditions

Those specific three conditions were chosen because they are:

- **Necessary**, otherwise one can write terms that don’t have any interpretation in the quantum world;
- **Sufficient** to obtain the soundness of the BC logic;
- **Simple and syntactic**, so we were able to integrate them inside Squirrel with a few hundred lines of code, only at the cost of a small expressivity loss.
**In a nut: an interactive prover for the BC logic**

- Relies on a meta-logic to allow for *mechanized proofs of protocol* for an unbounded number of sessions;
- gives *computational guarantees*;
- appears to be usable, and slowly starting to scale to more and more complex protocols.
What is Squirrel

In a nut: an interactive prover for the BC logic

- Relies on a meta-logic to allow for mechanized proofs of protocol for an unbounded number of sessions;
- gives computational guarantees;
- appears to be usable, and slowly starting to scale to more and more complex protocols.

Some figures

- 5 people core team: David Baelde, Stéphanie Delaune, Charlie J., Adrien Koutsos, Solène Moreau (and expanding)
- 30 000 lines of code and celebrating our 2 000 commit!
- about 15 real life case studies of protocols
## Implementation and Case-studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>LoC</th>
<th>Assumptions</th>
<th>Security properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key exchange protocols</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IkeV1&lt;sub&gt;psk&lt;/sub&gt;</td>
<td>680</td>
<td>PRF, EUF-CMA</td>
<td>Strong Secrecy &amp; Authentication</td>
</tr>
<tr>
<td>IkeV2&lt;sub&gt;sign&lt;/sub&gt;</td>
<td>300</td>
<td>PRF, EUF-CMA</td>
<td>Strong Secrecy &amp; Authentication</td>
</tr>
<tr>
<td>KE&lt;sub&gt;BCGNP&lt;/sub&gt;</td>
<td>355</td>
<td>PRF, IND-CCA, XOR</td>
<td>Strong Secrecy &amp; Implicit Authentication</td>
</tr>
<tr>
<td>KE&lt;sub&gt;FSXY&lt;/sub&gt;</td>
<td>660</td>
<td>PRF, IND-CCA, XOR</td>
<td>Strong Secrecy &amp; Implicit Authentication</td>
</tr>
<tr>
<td>SC-AKE</td>
<td>650</td>
<td>PRF, IND-CCA, SUF-CMA, XOR</td>
<td>Strong Secrecy &amp; Authentication</td>
</tr>
<tr>
<td><strong>Proving post-quantum soundness of Squirrel</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic Hash</td>
<td>100</td>
<td>PRF, EUF-CMA</td>
<td>Authentication &amp; Unlinkability</td>
</tr>
<tr>
<td>Hash Lock</td>
<td>130</td>
<td>PRF, EUF-CMA</td>
<td>Authentication &amp; Unlinkability</td>
</tr>
<tr>
<td>LAK (with pairs)</td>
<td>250</td>
<td>PRF, EUF-CMA</td>
<td>Authentication &amp; Unlinkability</td>
</tr>
<tr>
<td>MW</td>
<td>300</td>
<td>PRF, EUF-CMA, XOR</td>
<td>Authentication &amp; Unlinkability</td>
</tr>
<tr>
<td>Feldhofer</td>
<td>270</td>
<td>ENC-KP, INT-CTXT</td>
<td>Authentication &amp; Unlinkability</td>
</tr>
<tr>
<td>Private Authentication</td>
<td>100</td>
<td>IND-CCA, ENC-KP</td>
<td>Anonymity</td>
</tr>
</tbody>
</table>
What’s next?
Recap

Our contribution

The first interactive protocol prover that also provides post-quantum guarantees.
## Recap

### Our contribution

The first **interactive protocol prover** that also provides **post-quantum guarantees**.

### Limitation

Our key-exchange case-study do not cover any complex properties or compromise model, and there are no clear framework to prove key-exchanges in Squirrel.
### Recap

<table>
<thead>
<tr>
<th><strong>Our contribution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The first <em>interactive protocol prover</em> that also provides <em>post-quantum guarantees</em>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Limitation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Our key-exchange case-study do not cover any complex properties or compromise model, and there are no clear framework to prove key-exchanges in Squirrel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Natural next step</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Foundations for proving Key-Exchanges in Squirrel:</strong></td>
</tr>
<tr>
<td>Define how to express complex properties such as PFS or PCS in Squirrel, and simplified with our composition result.</td>
</tr>
<tr>
<td>Link proofs in Squirrel with existing framework (BR, CK,eCK, ...).</td>
</tr>
<tr>
<td>Perform an extensive case-study (KEMTLS).</td>
</tr>
</tbody>
</table>
The landscape

What we now have (thanks to 40 years of research!)
Many tools, attacker models and associated proof techniques. For instance:

- Proverif and Tamarin to verify at a high-level full protocol specifications;
- Squirrel to verify precisely the core of a protocol.
The landscape

What we now have (thanks to 40 years of research!)
Many tools, attacker models and associated proof techniques. For instance:

- Proverif and Tamarin to verify at a high-level full protocol specifications;
- Squirrel to verify precisely the core of a protocol.

But...  

- the tools are developed by distinct groups, and protocol analysis papers are often at a single level using a single tool;
- so many different approaches makes it difficult to export the tools and attacker models outside the protocol community.
The landscape

What we now have (thanks to 40 years of research!)
Many tools, attacker models and associated proof techniques. For instance:

- Proverif and Tamarin to verify at a high-level full protocol specifications;
- Squirrel to verify precisely the core of a protocol.

But...
- the tools are developed by distinct groups, and protocol analysis papers are often at a single level using a single tool;
- so many different approaches makes it difficult to export the tools and attacker models outside the protocol community.

Our goal
Build bridges inside the different groups in the community, as well as outside the community.
Medium-term goal

**Issue**

The existing protocol analysis papers are often at a single level using a single tool.
Medium-term goal

**Issue**
The existing protocol analysis papers are often at a single level using a single tool.

**Goal**
Design a platform that allows to formally combine the guarantees of multiple tools:
### Medium-term goal

#### Issue
The existing protocol analysis papers are often at a single level using a single tool.

#### Goal
Design a platform that allows to formally combine the guarantees of multiple tools:

- First for Tamarin and Proverif, to combine their complementary strengths;
## Medium-term goal

### Issue
The existing protocol analysis papers are often at a single level using a single tool.

### Goal
Design a platform that allows to formally combine the guarantees of multiple tools:
- First for Tamarin and Proverif, to combine their complementary strengths;
- Then integrate Squirrel;
Medium-term goal

**Issue**
The existing protocol analysis paper are often at a single level using a single tool.

**Goal**
Design a platform that allows to formally combine the guarantees of multiple tool:

- First for Tamarin and Proverif, to combine there complementary strengths;
- Then integrate Squirrel;
- Make a concrete multi-level analysis.
Formal Methods

One tool to use them all, and formally combine guarantees

Cryptographers

Use the tools straight away in new protocol designs

Standardizations

Provide all standards with formal models

Participate in the development of new standards

Companies

Governments

Make some techniques available to non-experts

Beyond protocols?

attacker models for code level analysis, e.g. for fault-injection
Long term goal

Formal Methods

One tool to use them all, and formally combine guarantees
Long term goal

Formal Methods

One tool to use them all, and formally combine guarantees

Cryptographers

Beyond protocols? Attacker models for code level analysis, e.g. for fault-injection
Long term goal

Formal Methods
One tool to use them all, and formally combine guarantees

Cryptographers
Use the tools straight away in new protocol designs

- Standardizations
  - Provide all standards with formal models
  - Participate in the development of new standards

- Companies
  - Governments
  - Make some techniques available to non-experts

Beyond protocols? 
Attacker models for code level analysis, e.g. for fault-injection
Long term goal

Formal Methods

One tool to use them all, and formally combine guarantees

Cryptographers

Use the tools straight away in new protocol designs

Standardizations

Companies
Governments
Make some techniques available to non-experts

Beyond protocols? attacker models for code level analysis, e.g. for fault-injection
Long term goal

- Use the tools straight away in new protocol designs
- Provide all standards with formal models
- Participate in the development of new standards

Formal Methods

One tool to use them all, and formally combine guarantees

Cryptographers

Standardizations

- Companies
- Governments

Make some techniques available to non-experts

Beyond protocols? Attacker models for code level analysis, e.g. for fault-injection
Long term goal

Formal Methods

One tool to use them all, and formally combine guarantees

- Use the tools straight away in new protocol designs
- Provide all standards with formal models
- Participate in the development of new standards
**Long term goal**

**Formal Methods**

- Use the tools straight away in new protocol designs
- Provide all standards with formal models
- Participate in the development of new standards

**Companies**

- Make some techniques available to non-experts

**Governments**

**Standardizations**

- One tool to use them all, and formally combine guarantees

**Cryptographers**

- Use the tools straight away in new protocol designs

Beyond protocols? Attacker models for code level analysis, e.g. for fault-injection
Long term goal

One tool to use them all, and formally combine guarantees

- Use the tools straight away in new protocol designs
- Provide all standards with formal models
- Participate in the development of new standards

- Make some techniques available to non-experts

Beyond protocols?

Cryptographers

Companies Governments

Standardizations

Formal Methods
Long term goal

- attacker models for code level analysis, e.g. for fault-injection
- Make some techniques available to non-experts
- Use the tools straight away in new protocol designs
- Provide all standards with formal models
- Participate in the development of new standards

Formal Methods

One tool to use them all, and formally combine guarantees