Exploring post-quantum cryptographic algorithms in Cryptol

Luís Miguel Pereira Constantino Romano, pg25311
Vitor Manuel Parreira Pereira, pg25301

DI - UM

MFES Milestone 4 - July 2, 2014

Outline

1. Recalling last Milestones
2. Properties proved
   - Matrix module
   - Hash functions
   - LWE public key encryption scheme
   - Lyubashevsky and Micciancio digital signature scheme
   - Lapin authentication protocol
3. Summing up
4. Conclusions/Future Work

Outline

1. Recalling last Milestones
2. Properties proved
   - Matrix module
   - Hash functions
   - LWE public key encryption scheme
   - Lyubashevsky and Micciancio digital signature scheme
   - Lapin authentication protocol
3. Summing up
4. Conclusions/Future Work

Milestone I

Milestone I

- Studied post-quantum cryptographic primitives:
  - LWE encryption scheme;
  - SWIFFT hash function;
  - Lyubashevsky and Micciancio digital signature scheme;
  - Lapin authentication protocol.

Milestone I

- Studied post-quantum cryptographic primitives:
  - LWE encryption scheme;
  - SWIFFT hash function;
  - Lyubashevsky and Micciancio digital signature scheme;
  - Lapin authentication protocol.
- Started exploring the Cryptol language:
  - Implemented some simple encryption schemes;
  - Tested the Cryptol prelude;
  - Tested the *:sat*, *:check* and *:prove* commands.

Milestone II

Milestone II

Specifications:

Milestone II

Specifications:

- Specification of the LWE encryption scheme;

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

As future work:

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

As future work:

- Study Cryptol support for polynomials;

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

As future work:

- Study Cryptol support for polynomials;
- Specify the Lapin authentication protocol;

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

As future work:

- Study Cryptol support for polynomials;
- Specify the Lapin authentication protocol;
- Review the implementation of the digital signature scheme;

Milestone II

Specifications:

- Specification of the LWE encryption scheme;
- Specification of the SWIFFT hash function;
- Specification of the Lyubashevsky and Micciancio digital signature scheme.

As future work:

- Study Cryptol support for polynomials;
- Specify the Lapin authentication protocol;
- Review the implementation of the digital signature scheme;
- Prove some properties about the primitives using Cryptol.

Milestone III

Milestone III

In addition to what we already had:

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;
- Specification of the Ajtai's hash function;

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;
- Specification of the Ajtai's hash function;
- Specification of the ideal lattices hash function;

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;
- Specification of the Ajtai's hash function;
- Specification of the ideal lattices hash function;
- *Checked* some properties about our specifications.

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;
- Specification of the Ajtai's hash function;
- Specification of the ideal lattices hash function;
- *Checked* some properties about our specifications.

Missing:

Milestone III

In addition to what we already had:

- Specification of the Lapin authentication protocol;
- Specification of the Ajtai's hash function;
- Specification of the ideal lattices hash function;
- *Checked* some properties about our specifications.

Missing:

- *Check* more properties about our specifications.

For this Milestone

For this Milestone

- Review the specifications;

For this Milestone

- Review the specifications;
- Prove some properties about the specifications;

For this Milestone

- Review the specifications;
- Prove some properties about the specifications;
- Comment the specifications;

For this Milestone

- Review the specifications;
- Prove some properties about the specifications;
- Comment the specifications;
- Write a report.

Outline

What to prove?

What to prove?

Finished the specifications,

What to prove?

Finished the specifications,

Prove that the specifications match the desired properties

↓

Cryptol high assurance programming

What to prove?

Finished the specifications,

Prove that the specifications match the desired properties

↓

Cryptol high assurance programming

Where to prove properties?

What to prove?

Finished the specifications,

Prove that the specifications match the desired properties

↓

Cryptol high assurance programming

Where to prove properties?

- Matrix module;
- Hash functions;
- Encryption scheme;
- Digital signature scheme;
- Authentication protocol.

Properties proved in the matrix module

Properties proved in the matrix module

- Matrix addition commutativity;
- Matrix addition associativity;
- Matrix addition identity element;
- Matrix multiplication associativity;
- Matrix multiplication right distributivity;
- Matrix multiplication left distributivity;
- Matrix multiplication zero element;
- Matrix modulo operation results on a new matrix where every element is lower that the modulo and greater than zero;
- Applying the inverse of a function after the original function results in the original input.

Properties proved in the hash functions

Properties proved in the hash functions

Collision resistance property:

Properties proved in the hash functions

Collision resistance property:

```
property ColisionResistance k m0 m1 =
     if m0 == m1 then (hash k m0) == (hash k m1)
     else (hash k m0) != (hash k m1)
```

Properties proved in the hash functions

Collision resistance property:

```
property ColisionResistance k m0 m1 =
     if m0 == m1 then (hash k m0) == (hash k m1)
     else (hash k m0) != (hash k m1)
```

:prove universally quantifies k, $m0$ and $m1$ and runs all possible inputs

Properties proved in the hash functions

Collision resistance property:

```
property ColisionResistance k m0 m1 =
     if m0 == m1 then (hash k m0) == (hash k m1)
     else (hash k m0) != (hash k m1)
```

:prove universally quantifies k, $m0$ and $m1$ and runs all possible inputs

↓

It would always find a collision (in exponential time)

Properties proved in the hash functions

Solution is to use the *:check* command!

Properties proved in the hash functions

Solution is to use the *:check* command!

Run *:check* command polynomial times

Properties proved in the hash functions

Solution is to use the *:check* command!

Run *:check* command polynomial times

↓

No collision found!

Properties proved in the hash functions

Solution is to use the *:check* command!

Run *:check* command polynomial times

$\downarrow$

No collision found!

$\downarrow$

Hash function is collision resistant

Properties proved in the hash functions

Solution is to use the *:check* command!

Run *:check* command polynomial times

↓

No collision found!

↓

Hash function is collision resistant

This holds for all the hash functions specified.

Properties proved in the LWE encryption scheme

Properties proved in the LWE encryption scheme

Remember encryption and decryption algorithm:

Properties proved in the LWE encryption scheme

Remember encryption and decryption algorithm:

Encryption: *Given a message* $v \in \mathbb{Z}_t^l$ *and a public key (A, P), choose a random vector* $a = \{-r, -r+1, ...r\}^m$ *and output the ciphertext* $(u = A^T a, c = P^T a + \underline{f}(v))$

Decryption: *Given a ciphertext (u, c) and a private key S output* $\underline{f^{-1}}(c - S^T u)$.

Properties proved in the LWE encryption scheme

Remember encryption and decryption algorithm:
Encryption: Given a message $v \in \mathbb{Z}_t^l$ and a public key (A, P), choose a random vector $a = \{-r, -r+1, ...r\}^m$ and output the ciphertext $(u = A^T a$, $c = P^T a + \underline{f}(v))$
Decryption: Given a ciphertext (u, c) and a private key S output $\underline{f^{-1}}(c - S^T u)$.

f^{-1} must be the inverse of f

Properties proved in the LWE encryption scheme

Remember encryption and decryption algorithm:
Encryption: Given a message $v \in \mathbb{Z}_t^l$ *and a public key* (A, P)*, choose a random vector* $a = \{-r, -r+1, ...r\}^m$ *and output the ciphertext* $(u = A^T a, c = P^T a + \underline{f}(v))$
Decryption: Given a ciphertext (u, c) *and a private key* S *output* $\underline{f^{-1}}(c - S^T u)$.

f^{-1} must be the inverse of f

$\downarrow$

Property proved!

Properties proved in the LWE encryption scheme

Properties proved in the LWE encryption scheme

Correction property:

Properties proved in the LWE encryption scheme

Correction property:

```
property CorrectScheme m =
  decrypt keyGen.1 (encrypt keyGen.2 m) == m
```

Properties proved in the LWE encryption scheme

Correction property:

```
property CorrectScheme m =
  decrypt keyGen.1 (encrypt keyGen.2 m) == m
```

Can not run the *:prove* command due to space requirements!

Properties proved in the LWE encryption scheme

Correction property:

```
property CorrectScheme m =
  decrypt keyGen.1 (encrypt keyGen.2 m) == m
```

Can not run the *:prove* command due to space requirements!

↓

And, since there is decryption error probability it would not prove!

Properties proved in the LWE encryption scheme

Correction property:

```
property CorrectScheme m =
  decrypt keyGen.1 (encrypt keyGen.2 m) == m
```

Can not run the *:prove* command due to space requirements!

↓

And, since there is decryption error probability it would not prove!

↓

Solution: run the *:check* command

Problems with Cryptol

Problems with Cryptol

Recalling Milestone II:

Problems with Cryptol

Recalling Milestone II:

Also, impossible to load matrix E from LWE cryptosystem to Cryptol prelude.

Problems with Cryptol

Recalling Milestone II:

Also, impossible to load matrix E from LWE cryptosystem to Cryptol prelude.

Cryptol prelude takes too long to load matrix E (1319×166) to the prelude (Cryptol bug?)

Problems with Cryptol

Recalling Milestone II:

Also, impossible to load matrix E from LWE cryptosystem to Cryptol prelude.

Cryptol prelude takes too long to load matrix E (1319×166) to the prelude (Cryptol bug?)

$\downarrow$

Due to inefficient problems

Problems with Cryptol

Recalling Milestone II:

Also, impossible to load matrix E from LWE cryptosystem to Cryptol prelude.

Cryptol prelude takes too long to load matrix E (1319×166) to the prelude (Cryptol bug?)

$\downarrow$

Due to inefficient problems

$\downarrow$

Not able to *:check* the property

Properties proved in the digital signature scheme

Properties proved in the digital signature scheme

Correction property:

Properties proved in the digital signature scheme

Correction property:

```
property Correction msg =
  verify keyGenDS.2 msg (sign keyGenDS.1 msg) == True
```

Properties proved in the digital signature scheme

Correction property:

```
property Correction msg =
  verify keyGenDS.2 msg (sign keyGenDS.1 msg) == True
```

Can not run the *:prove* command due to space requirements!

Properties proved in the digital signature scheme

Correction property:

```
property Correction msg =
  verify keyGenDS.2 msg (sign keyGenDS.1 msg) == True
```

Can not run the *:prove* command due to space requirements!

↓

Solution: run the *:check* command

Problems with Cryptol

Problems with Cryptol

Efficiency problems (don't know why)

Problems with Cryptol

Efficiency problems (don't know why)

$\downarrow$

A single test takes a very long time to perform

Problems with Cryptol

Efficiency problems (don't know why)

$\downarrow$

A single test takes a very long time to perform

$\downarrow$

:check command takes a really long time to execute

Problems with Cryptol

Efficiency problems (don't know why)

$\downarrow$

A single test takes a very long time to perform

$\downarrow$

:check command takes a really long time to execute

$\downarrow$

Don't have any result for this specification

Properties proved in the Lapin authentication protocol

Properties proved in the Lapin authentication protocol

Correction property:

Properties proved in the Lapin authentication protocol

Correction property:

```
property Correction c = (reader r (tag c) == True)
```

Properties proved in the Lapin authentication protocol

Correction property:

```
property Correction c = (reader r (tag c) == True)
```

For Lapin, the input space allows us to test all the possible inputs!

However, some problems arose when trying to prove correctness

Problems with Cryptol

Problems with Cryptol

Recalling Milestone III:

Problems with Cryptol

Recalling Milestone III:

Having some problems with the Lapin specification - we are not able to use :check with polymorphic types.

Problems with Cryptol

Recalling Milestone III:

Having some problems with the Lapin specification - we are not able to use :check with polymorphic types.

Solution $\rightarrow$ Use Cryptol inferred types

Problems with Cryptol

Recalling Milestone III:

Having some problems with the Lapin specification - we are not able to use :check with polymorphic types.

Solution $\rightarrow$ Use Cryptol inferred types

However,

Problems with Cryptol

Recalling Milestone III:

Having some problems with the Lapin specification - we are not able to use :check with polymorphic types.

Solution → Use Cryptol inferred types

However,

- Cryptol does no agree with its own inferred types! (Crytpol bug?)

Problems with Cryptol

Recalling Milestone III:

Having some problems with the Lapin specification - we are not able to use :check with polymorphic types.

Solution → Use Cryptol inferred types

However,

- Cryptol does no agree with its own inferred types! (Crytpol bug?)
- Not able to *:prove* or *:check* the property

Outline

Galois' proposal - Exploring post-quantum cryptographic algorithms in Cryptol

Galois' proposal - Exploring post-quantum cryptographic algorithms in Cryptol

If quantum computers prove successful, and are able to scale sufficiently to apply Shor's algorithm to factoring products of large primes, then the public key cryptosystems that rely on the difficulty of such problems will effectively be broken. This project would involve getting familiar with, and implementing in Cryptol, any of a variety of algorithms that would be secure in a post-quantum future. For references, see the wikipedia articles on "Post-quantum cryptography", "Lattice-based cryptography," "Multivariate cryptography," or explore other public key cryptosystems that meet this requirement. The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol.

Objectives traced...

Objectives traced...

- What is post-quantum cryptography?

Objectives traced...

- What is post-quantum cryptography?
- Study post-quantum cryptographic primitives;

Objectives traced...

- What is post-quantum cryptography?
- Study post-quantum cryptographic primitives;
- Get involved with the Cryptol language;

Objectives traced...

- What is post-quantum cryptography?
- Study post-quantum cryptographic primitives;
- Get involved with the Cryptol language;
- Specify the studied primitives in Cryptol;

Objectives traced...

- What is post-quantum cryptography?
- Study post-quantum cryptographic primitives;
- Get involved with the Cryptol language;
- Specify the studied primitives in Cryptol;
- Try to prove some properties about those specifications.

Objectives completed

Objectives completed

- What is post-quantum cryptography? ✓
- Study post-quantum cryptographic primitives; ✓
- Get involved with the Cryptol language; ✓
- Specify the studied primitives in Cryptol; ✓
- Try to prove some properties about those specifications. ✓

What do we present?

What do we present?

Cryptol specifications:

What do we present?

Cryptol specifications:

- Matrix module;
- Ajtai hash function;
- Ideal lattices based hash function;
- SWIFFT hash function;
- LWE public key encryption scheme;
- Lyubashevsky and Micciancio digital signature scheme;
- Lapin authentication protocol.

Outline

1. Recalling last Milestones
2. Properties proved
   - Matrix module
   - Hash functions
   - LWE public key encryption scheme
   - Lyubashevsky and Micciancio digital signature scheme
   - Lapin authentication protocol
3. Summing up
4. Conclusions/Future Work

Conclusions

Conclusions

Quantum computation is still something unachieved. However, it is almost consensual that quantum computers will be a reality in the future

Conclusions

Quantum computation is still something unachieved. However, it is almost consensual that quantum computers will be a reality in the future

Quantum computation will force computer scientists to review modern computation techniques

Conclusions

Quantum computation is still something unachieved. However, it is almost consensual that quantum computers will be a reality in the future

Quantum computation will force computer scientists to review modern computation techniques

Quantum computation allows the design of algorithms that efficiently solve hard mathematical problems

Conclusions

Quantum computation is still something unachieved. However, it is almost consensual that quantum computers will be a reality in the future

Quantum computation will force computer scientists to review modern computation techniques

Quantum computation allows the design of algorithms that efficiently solve hard mathematical problems

Specifications of cryptographic primitives is a very interesting way to see their behaviour and to explore them, representing a good alternative to code verification

Future Work

Future Work

When code generation is available for 2.0, generate code from the specifications and deploy a post-quantum cryptography library

Future Work

When code generation is available for 2.0, generate code from the specifications and deploy a post-quantum cryptography library

Generate hardware oriented code and see how it behaves with algorithms related to number theory

Future Work

When code generation is available for 2.0, generate code from the specifications and deploy a post-quantum cryptography library

Generate hardware oriented code and see how it behaves with algorithms related to number theory

Implement this cryptographic primitives in some programming language and use the Galois Software Analysis Workbench (SAW) in order to prove that the program matches its specification

Answering Galois' question

Answering Galois' question

From Galois' proposal:

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

So, what makes an algorithm "quantum-safe"?

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

So, what makes an algorithm "quantum-safe"?

- Implemented algorithms rely on hard mathematical problems;

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

So, what makes an algorithm "quantum-safe"?

- Implemented algorithms rely on hard mathematical problems;
- No quantum efficient attack must be known to this problems;

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

So, what makes an algorithm "quantum-safe"?

- Implemented algorithms rely on hard mathematical problems;
- No quantum efficient attack must be known to this problems;
- One can never say they are impossible to solve. They simply believed to be hard problems since no solutions are known for them;

Answering Galois' question

From Galois' proposal:

The completed project would be an analysis of what makes a cryptographic algorithm "quantum safe", as well as the implementation in Cryptol

So, what makes an algorithm "quantum-safe"?

- Implemented algorithms rely on hard mathematical problems;
- No quantum efficient attack must be known to this problems;
- One can never say they are impossible to solve. They simply believed to be hard problems since no solutions are known for them;
- For ciphers based on ad-hoc principles (block ciphers), no quantum attack is known... But remember to use larger encryption keys!

Thank you for your attention

Exploring post-quantum cryptographic algorithms in Cryptol

Luís Miguel Pereira Constantino Romano, pg25311
Vitor Manuel Parreira Pereira, pg25301

DI - UM

MFES Milestone 4 - July 2, 2014