REDOCS'24

Designing a metamorphic test program in Rust for cryptographic primitives

Marin FRANCOIS [1], Tom GOUVILLE [2], Medhi KERMAOUI [2], Mahreen KHAN [3], Thomas PREVOST [4]

[1]LAMSADE, CNRS, Université Paris-Dauphine PSL

[2]LORIA, INRIA, Université de Lorraine

[3]COMELEC, Télécom Paris, Institut Polytechnique de Paris

[4]I3S, CNRS, Université Côte d'Azur

“It is not a bug, just an interesting unexpected behavior”

extrapolated from Fenzi *et al.*, 2018

Outline

- Research Objectives
- Background
  - Cryptographic Primitives
  - Security Requirements
  - Testing Crypto Requirements
- RQ0. Metamorphic testing
  - Metamorphic Testing Recipe
  - Practical Example : Kyber KEM
  - Experimentation and Results
- RQ1. Experimentation Details
  - Metamorphic Tests
  - Statistical Tests
- RQ2. Writing a Rust library
  - Introduction to Rust
  - Rust Crypto
  - Our library : `metamorphic-testing-rs`
- Conclusions
  - Key Takeaways

Research Objectives

1. **RQ0.** Review of a broad set of cryptographic primitives
2. **RQ1.** Build new metamorphic testing protocols
3. **RQ2.** Test *Rust Crypto* [1] for bugs

[1] https://github.com/RustCrypto

Outline

Background

Cryptographic Primitives

Security Requirements

Testing Crypto Requirements

Cryptographic Primitives Principles

- **Building blocks** for any cryptographic protocol

Cryptographic Primitives Principles

- **Building blocks** for any cryptographic protocol
- **Ensure** confidentiality, authentication, integrity

Cryptographic Primitives Principles

- **Building blocks** for any cryptographic protocol
- **Ensure** confidentiality, authentication, integrity
- **Specific Properties** are required to be valid

Cryptographic Primitives Principles

- **Building blocks** for any cryptographic protocol
- **Ensure** confidentiality, authentication, integrity
- **Specific Properties** are required to be valid

Cryptographic Primitives Examples

- Hash Functions (*e.g.*, SHA-256)

Cryptographic Primitives Examples

- Hash Functions (*e.g.*, SHA-256)
- Key Exchange Mechanism (KEM) (*e.g.* Kyber)

Cryptographic Primitives Examples

- Hash Functions (*e.g.*, SHA-256)
- Key Exchange Mechanism (KEM) (*e.g.* Kyber)
- Symmetric Encryption (SE) (*e.g.*, AES)

Cryptographic Primitives Examples

- Hash Functions (*e.g.*, SHA-256)
- Key Exchange Mechanism (KEM) (*e.g.* Kyber)
- Symmetric Encryption (SE) (*e.g.*, AES)
- Message Authentication Codes (*e.g.*, HMAC-SHA256)

Outline

Background

Cryptographic Primitives

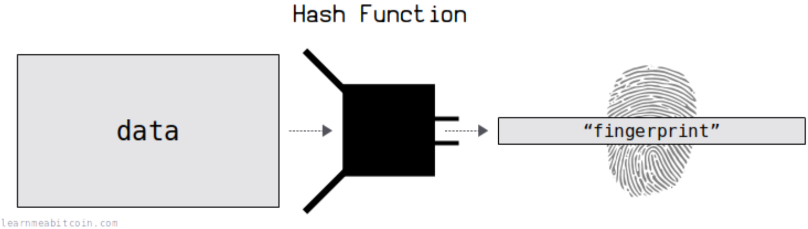
Security Requirements

Testing Crypto Requirements

What are security requirements ?

Explicit and **necessary** properties of crypto primitives

Hash Functions



Hash Function Security Reqs.

Hash Security Reqs :

1. First preimage resistance
   i.e. Cannot retrieve x from $H(x)$

Hash Function Security Reqs.

Hash Security Reqs :

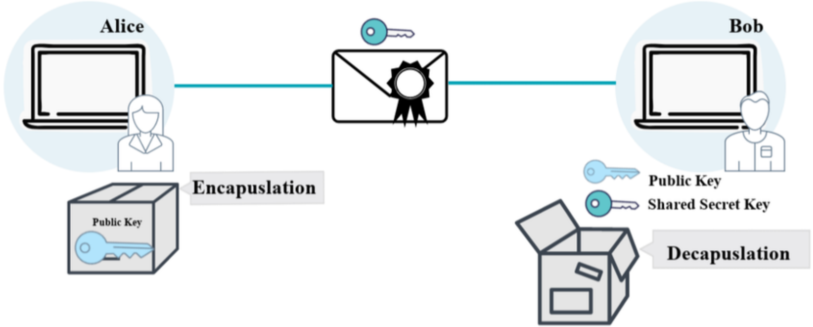
1. First preimage resistance
   i.e. Cannot retrieve x from $H(x)$
2. Second preimage resistance
   i.e. Cannot retrieve x' from $H(x)$ s.t. $H(x') = H(x)$

Hash Function Security Reqs.

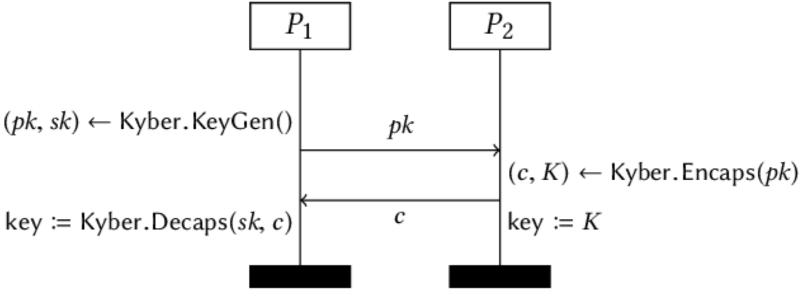
Hash Security Reqs :

1. First preimage resistance
   i.e. Cannot retrieve x from $H(x)$
2. Second preimage resistance
   i.e. Cannot retrieve x' from $H(x)$ s.t. $H(x') = H(x)$
3. Collision resistance
   i.e. Cannot retrieve (x, y) s.t. $H(x) = H(y)$

Key Encapsulation Mechanisms (KEM)



Key Encapsulation Mechanisms (KEM) (2)



P_1

P_2

$(pk, sk) \leftarrow \text{Kyber.KeyGen}()$

pk

$(c, K) \leftarrow \text{Kyber.Encaps}(pk)$

c

$\text{key} := \text{Kyber.Decaps}(sk, c)$

$\text{key} := K$

KEM Security Requirements

1. IND-CPA
   i.e. Indistinguishability of secrets under Chosen Plaintext Attack

KEM Security Requirements

1. IND-CPA
   i.e. Indistinguishability of secrets under Chosen Plaintext Attack

2. IND-CCA
   i.e. Indistinguishability of secrets under Chosen Ciphertext Attack

KEM Security Requirements

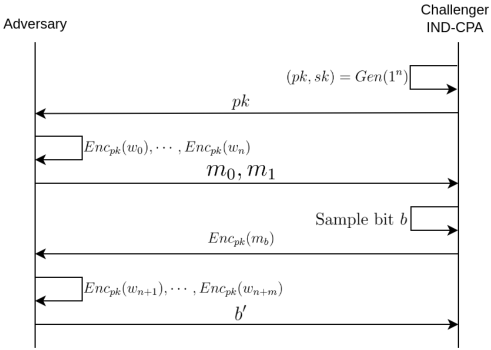
1. IND-CPA
   i.e. Indistinguishability of secrets under Chosen Plaintext Attack

2. IND-CCA
   i.e. Indistinguishability of secrets under Chosen Ciphertext Attack

3. IND-qCCA
   i.e. Indistinguishability of secrets under Chosen Ciphertext Attack
   with Quantum Computer

KEM Security Requirements

1. IND-CPA
   i.e. Indistinguishability of secrets under Chosen Plaintext Attack
2. IND-CCA
   i.e. Indistinguishability of secrets under Chosen Ciphertext Attack
3. IND-qCCA
   i.e. Indistinguishability of secrets under Chosen Ciphertext Attack with Quantum Computer



Outline

Background

- Cryptographic Primitives
- Security Requirements
- Testing Crypto Requirements

Testing security requirements (old way)

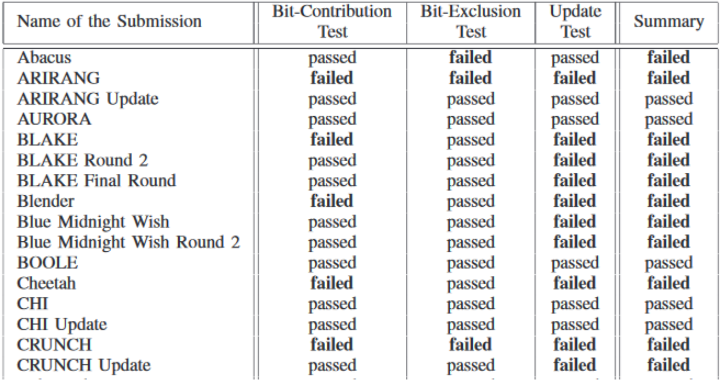
Algo. testing using tests vectors

```
**Input:** $p \leftarrow \{0,1\}^l$ // *Test vector input*
**Input:** $h_T \leftarrow SHA3_{REF}(p)$ // *Test vector ouput*
**Output:** Ok:pass

$h_t \leftarrow SHA3_{TEST}(p)$
$\texttt{assert}(h_t == h_T)$ // *Compare to Ref.*
```

cosmian REDOCS

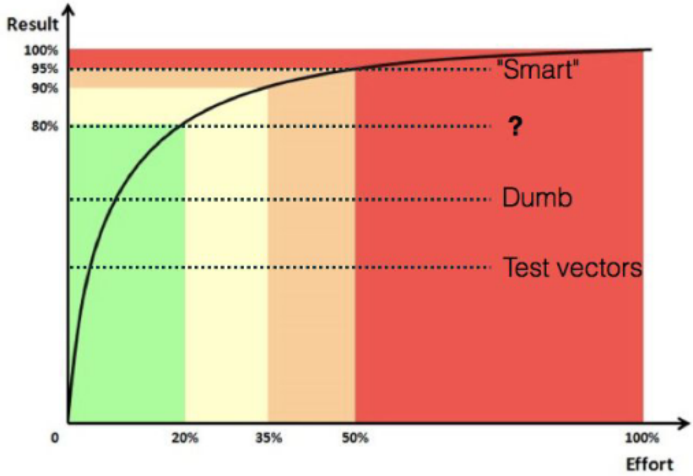
Just *trust* the reference implementation ... or don't ?



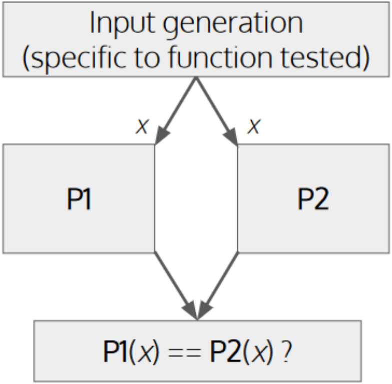
Name of the Submission	Bit-Contribution Test	Bit-Exclusion Test	Update Test	Summary
Abacus	passed	**failed**	passed	**failed**
ARIRANG	**failed**	**failed**	**failed**	**failed**
ARIRANG Update	passed	passed	passed	passed
AURORA	passed	passed	passed	passed
BLAKE	**failed**	passed	**failed**	**failed**
BLAKE Round 2	passed	passed	**failed**	**failed**
BLAKE Final Round	passed	passed	**failed**	**failed**
Blender	**failed**	passed	**failed**	**failed**
Blue Midnight Wish	passed	passed	**failed**	**failed**
Blue Midnight Wish Round 2	passed	passed	**failed**	**failed**
BOOLE	passed	passed	passed	passed
Cheetah	**failed**	passed	**failed**	**failed**
CHI	passed	passed	passed	passed
CHI Update	passed	passed	passed	passed
CRUNCH	**failed**	**failed**	**failed**	**failed**
CRUNCH Update	passed	passed	**failed**	**failed**

Figure: *Post-mortem* testing of SHA-3 NIST reference implem. [Mou+18]

Differential Fuzzing with CDF (2017) (1)



Differential Fuzzing with CDF (2017) (2)



Differential Fuzzing with CDF (2017) (3)

Problem

What if you don't have two different implementations ?

Outline

RQ0. Metamorphic testing

- Metamorphic Testing Recipe
- Practical Example : Kyber KEM
- Experimentation and Results

Metamorphic Testing Recipe

1. **Identify Metamorphic Relations (MRs)**
   → Find properties relating input/output changes

Metamorphic Testing Recipe

1. **Identify Metamorphic Relations (MRs)**
   → Find properties relating input/output changes
2. **Generate Initial Test Cases**
   → Run the program with initial inputs

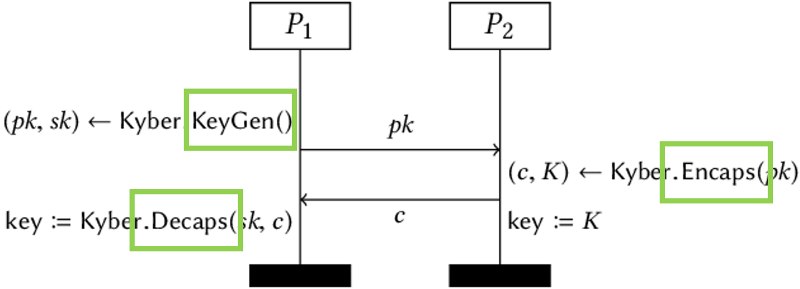
Metamorphic Testing Recipe

1. **Identify Metamorphic Relations (MRs)**
   → Find properties relating input/output changes
2. **Generate Initial Test Cases**
   → Run the program with initial inputs
3. **Maul Input**
   → Modify input based on MRs and generate new test cases.

Metamorphic Testing Recipe

1. **Identify Metamorphic Relations (MRs)**
   → Find properties relating input/output changes
2. **Generate Initial Test Cases**
   → Run the program with initial inputs
3. **Maul Input**
   → Modify input based on MRs and generate new test cases.
4. **Verify Outputs**
   → Ensure modified inputs produce expected results from MR.

Metamorphic Test Gadgets



P_1

P_2

$(pk, sk) \leftarrow \text{Kyber}.\boxed{\text{KeyGen}()}$

pk

$(c, K) \leftarrow \text{Kyber}.\boxed{\text{Encaps}(}pk)$

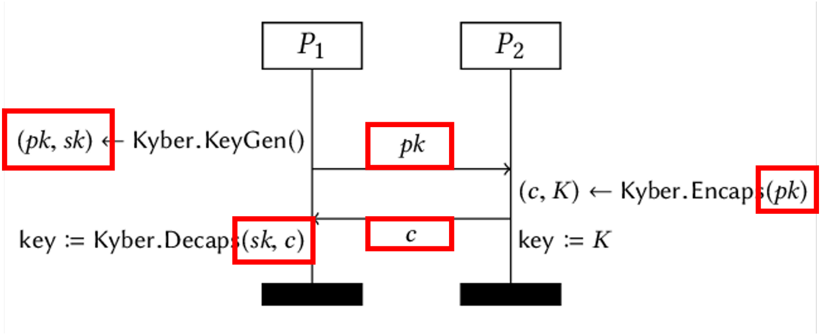
c

$\text{key} := \text{Kyber}.\boxed{\text{Decaps}(}sk, c)$

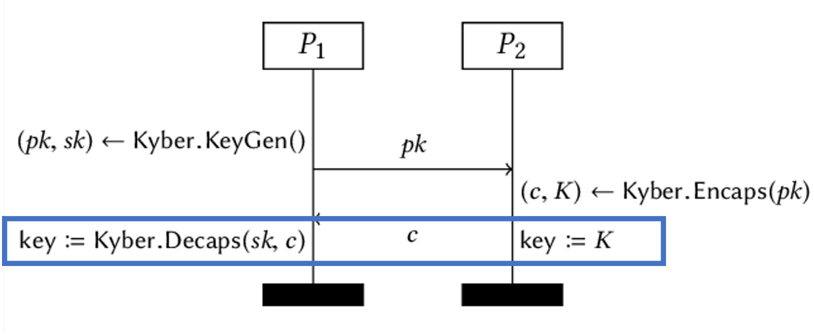
$\text{key} := K$

cosmian REDOCS

Metamorphic Test Inputs

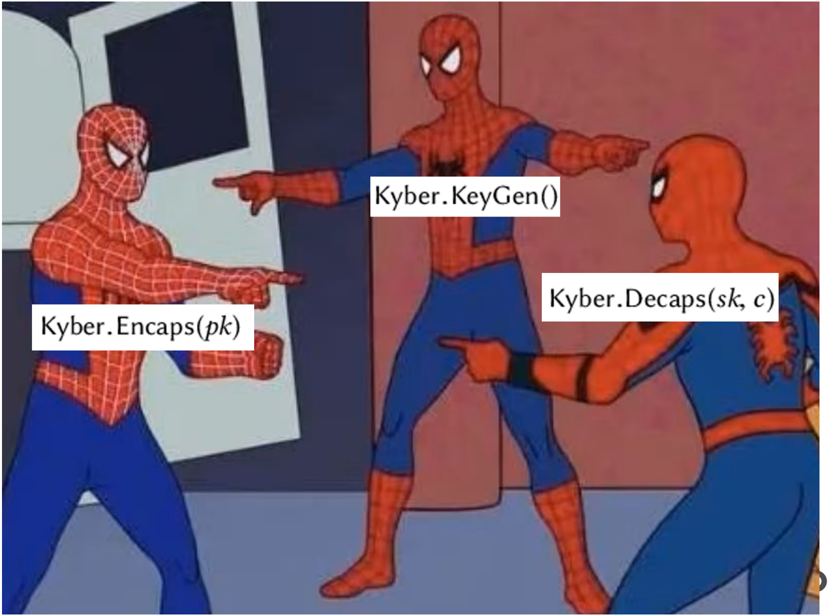

P_1
P_2
$(pk, sk) \leftarrow$ Kyber.KeyGen()
pk
$(c, K) \leftarrow$ Kyber.Encaps(pk)
c
key := Kyber.Decaps(sk, c)
key := K

Metamorphic Test MR



P_1
P_2
$(pk, sk) \leftarrow \texttt{Kyber.KeyGen}()$
pk
$(c, K) \leftarrow \texttt{Kyber.Encaps}(pk)$
$\text{key} := \texttt{Kyber.Decaps}(sk, c)$
c
$\text{key} := K$

MRs in a nutshell



Metamorphic Example

Target MR :

$$\texttt{Kyber} \quad : \quad \texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0, 1\}^l$

Metamorphic Example

Target MR :

$$\mathtt{Kyber} \quad : \quad \mathtt{encaps}(pk, r) = \mathtt{decaps}(sk, \mathtt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0, 1\}^{l}$
Equality: $(p_k, s_k) \leftarrow \mathtt{Gen}(r)$

Metamorphic Example

Target MR :

$$\texttt{Kyber} \quad : \quad \texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0,1\}^l$
Equality: $(p_k, s_k) \leftarrow \texttt{Gen}(r)$
Output: Ok:pass

Metamorphic Example

Target MR :

$$\texttt{Kyber} \quad : \quad \texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0,1\}^{l}$
Equality: $(p_k, s_k) \leftarrow \texttt{Gen}(r)$
Output: Ok:pass

$pk, sk \leftarrow \texttt{Gen}(r)$
$ss, c \leftarrow \texttt{Encaps}(pk)$

Metamorphic Example

Target MR :

$$\texttt{Kyber} \quad : \quad \texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0, 1\}^{l}$
Equality: $(p_k, s_k) \leftarrow \texttt{Gen}(r)$
Output: Ok:pass

```
$pk, sk \leftarrow \texttt{Gen}(r)$
$ss, c \leftarrow \texttt{Encaps}(pk)$
**for** $i \leq len(pk)$ **do**
    $pk' \leftarrow \texttt{Maul}(pk)$
    $ss', c' \leftarrow \texttt{Encaps}(pk')$
```

Metamorphic Example

Target MR :

$$\texttt{Kyber} \quad : \quad \texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$$

Associated Test :

Input: $r \leftarrow_{\$} \{0,1\}^l$
Equality: $(p_k, s_k) \leftarrow \texttt{Gen}(r)$
Output: Ok:pass

```
$pk, sk \leftarrow \texttt{Gen}(r)$
$ss, c \leftarrow \texttt{Encaps}(pk)$
for $i \leq len(pk)$ do
    $pk' \leftarrow \texttt{Maul}(pk)$
    $ss', c' \leftarrow \texttt{Encaps}(pk')$
    $\texttt{assert}((ss', \_)! = (ss, \_))$
end for
```

Examples `Maul()`

- **Bit-Contribution**
  → *verify all input bits impact output*

Examples `Maul()`

- **Bit-Contribution**
  → *verify all input bits impact output*

- **Bit-Exclusion**
  → *verify padding related functions*

Examples `Maul()`

- **Bit-Contribution**
  → *verify all input bits impact output*
- **Bit-Exclusion**
  → *verify padding related functions*
- **Update**
  → *verify `SHA3`-like compress & update cycles*

Bit Contribution `Maul()`

Skeleton

```
Input: $x$
for $i \leq l = len(x)$ do
    $x' \leftarrow \{0\}^{l-i} || \{1\}^i$
    call
end for
```

Bit Contribution `Maul()`

Skeleton

```
Input: $x$
for $i \leq l = len(x)$ do
    $x' \leftarrow \{0\}^{l-i} || \{1\}^{i}$
    call
end for
```

Test Hash Function

```
Input: $x$
$h \leftarrow \mathtt{SHA256}(x)$
for $i \leq l = len(x)$ do
    $x' \leftarrow \{0\}^{l-i} || \{1\}^{i}$
    $h' \leftarrow \mathtt{SHA256}(x')$
    $\mathtt{assert}(h' ! = h)$
end for
```

Bit Exclusion `Maul()`

Skeleton

```
Input: $x$
$l_1 \leftarrow \text{len}(x) \mod 8$
$l_2 \leftarrow \text{len}(x)$
for $i = l_2 - 1$ to $l_1$ do
    $x' \leftarrow \{x\}^{l_2} || \{0\}^i$
    call
end for
```

Bit Exclusion `Maul()`

Skeleton

```
Input: $x$
$l_1 \leftarrow \text{len}(x) \mod 8$
$l_2 \leftarrow \text{len}(x)$
for $i = l_2 - 1$ to $l_1$ do
    $x' \leftarrow \{x\}^{l_2} || \{0\}^{i}$
    call
end for
```

Test Hash Function

```
Input: $x$
$h \leftarrow \texttt{SHA256}(x)$
for $i \leq l = len(x)$ do
    $x' \leftarrow$ Maul()
    $h' \leftarrow \texttt{SHA256}(x')$
    assert$(h'! = h)$
end for
```

Bit Exclusion `Maul()`

Skeleton

```
Input: $x$
$l_1 \leftarrow \text{len}(x) \mod 8$
$l_2 \leftarrow \text{len}(x)$
for $i = l_2 - 1$ to $l_1$ do
    $x' \leftarrow \{x\}^{l_2} || \{0\}^i$
    call
end for
```

Test Hash Function

```
Input: $x$
$h \leftarrow \texttt{SHA256}(x)$
for $i \leq l = len(x)$ do
    $x' \leftarrow \texttt{Maul()}$
    $h' \leftarrow \texttt{SHA256}(x')$
    $\texttt{assert}(h'! = h)$
end for
```

Note : This is especially useful for C programs

Outline

RQ0. Metamorphic testing

Metamorphic Testing Recipe

Practical Example : Kyber KEM

Experimentation and Results

Practical Example : Kyber KEM

Input: $r \leftarrow \mathrm{PRNG}()$

Practical Example : Kyber KEM

Input: $r \leftarrow \mathsf{PRNG}()$
Equality: $\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$

Practical Example : Kyber KEM

Input: $r \leftarrow \mathrm{PRNG}()$
Equality: $\mathtt{encaps}(pk, r) = \mathtt{decaps}(sk, \mathtt{encaps}(pk, r))$
Output: ok:pass

Practical Example : Kyber KEM

Input: $r \leftarrow \mathrm{PRNG}()$
Equality: $\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$
Output: ok:pass

$(pk, sk) \leftarrow \texttt{Gen}(r)$
$(ss, c) \leftarrow \texttt{Encaps}(pk, r)$

Practical Example : Kyber KEM

Input: $r \leftarrow \text{PRNG}()$
Equality: $\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$
Output: ok:pass

```
$(pk, sk) \leftarrow \texttt{Gen}(r)$
$(ss, c) \leftarrow \texttt{Encaps}(pk, r)$

for i = 0 to len(pk) do
    $pk' \leftarrow \texttt{Maul}(pk)$
    $(ss', c') \leftarrow \texttt{Encaps}(pk')$
```

Practical Example : Kyber KEM

```
**Input:** $r \leftarrow \text{PRNG}()$
**Equality:** $\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$
**Output:** ok:pass

$(pk, sk) \leftarrow \texttt{Gen}(r)$
$(ss, c) \leftarrow \texttt{Encaps}(pk, r)$

**for** i = 0 **to** $\texttt{len}$(pk) **do**
    $pk' \leftarrow \texttt{Maul}(pk)$
    $(ss', c') \leftarrow \texttt{Encaps}(pk')$
    **assert** $(ss' \neq ss)$
**end for**
```

Outline

RQ0. Metamorphic testing

Metamorphic Testing Recipe

Practical Example : Kyber KEM

Experimentation and Results

Overview of conducted tests (1)

Total : 69 test rounds, 21 primitives tested, 1 *bug* found (twice)

Overview of conducted tests (1)

Total : 69 test rounds, 21 primitives tested, 1 *bug* found (twice)

Crate	MR	Maul()	API	Bugs
PQC_KYBER	$\mathtt{encaps}(pk, r) = \mathtt{decaps}(sk, \mathtt{encaps}(pk, r))$	$\mathtt{gen}(\mathtt{maul}(\mathbf{r}))$	No	0
		$\mathtt{encaps}(pk, \mathtt{maul}(r))$	Yes	0
		$\mathtt{encaps}(\mathtt{maul}(pk), r)$	Yes	0
		$\mathtt{decaps}(\mathtt{maul}(sk), r)$	Yes	1
		$\mathtt{decaps}(sk, \mathtt{maul}(r))$	Yes	0

Overview of conducted tests (1)

Total : 69 test rounds, 21 primitives tested, 1 *bug* found (twice)

Crate	MR	Maul()	API	Bugs
PQC_KYBER	$\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$	$\texttt{gen}(\texttt{maul}(\mathbf{r}))$	No	0
		$\texttt{encaps}(pk, \texttt{maul}(r)$	Yes	0
		$\texttt{encaps}(\texttt{maul}(pk), r)$	Yes	0
		$\texttt{decaps}(\texttt{maul}(sk), r)$	Yes	1
		$\texttt{decaps}(sk, \texttt{maul}(r))$	Yes	0
ML-KEM	$\texttt{encaps}(pk, r) = \texttt{decaps}(sk, \texttt{encaps}(pk, r))$	$\texttt{gen}(\texttt{maul}(\mathbf{r}))$	No	0
		$\texttt{encaps}(pk, \texttt{maul}(r)$	Yes	0
		$\texttt{encaps}(\texttt{maul}(pk), r)$	Yes	0
		$\texttt{decaps}(\texttt{maul}(sk), r)$	Yes	1
		$\texttt{decaps}(sk, \texttt{maul}(r))$	Yes	0

Overview of conducted tests (1)

Crate	MR	`Maul()`	API	Bugs
`SHA2`	$\mathtt{H}(x) \sim U(0, 2^{\mathbf{256}})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{\mathbf{512}})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_{\mathbf{1}}), \mathtt{H}(x_{\mathbf{2}}), \ldots)$	No	0

Overview of conducted tests (1)

Crate	MR	`Maul()`	API	Bugs
`SHA2`	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
`SHA3`	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0

Overview of conducted tests (1)

Crate	MR	`Maul()`	API	Bugs
SHA2	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
SHA3	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
PBKDF2	$\mathtt{PBKDF2}(x) \sim U(0, 2^{160})$	$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0

Overview of conducted tests (1)

Crate	MR	`Maul()`	API	Bugs
SHA2	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
SHA3	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
PBKDF2	$\mathtt{PBKDF2}(x) \sim U(0, 2^{160})$	$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0
SHA2 Compression	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
SHA3 Compression	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0

Overview of conducted tests (1)

Crate	MR	Maul()	API	Bugs
SHA2	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
SHA3	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \neq \mathtt{H}(x')$	$\mathtt{H}(x) = \mathtt{update}(\mathtt{H}(x_1), \mathtt{H}(x_2), \ldots)$	No	0
PBKDF2	$\mathtt{PBKDF2}(x) \sim U(0, 2^{160})$	$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA2}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0
		$\mathtt{pbkdf2}(\mathtt{SHA3}(\mathtt{maul}(x)))$	No	0
SHA2 Compression	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
SHA3 Compression	$\mathtt{H}(x) \sim U(0, 2^{256})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
	$\mathtt{H}(x) \sim U(0, 2^{512})$	$\mathtt{H}(\mathtt{maul}(x))$	No	0
BLAKE2	$\mathtt{H}(x) \sim U(0, 2^{k})$	$\mathtt{H}(\mathtt{maul}(x))$	Yes	0
BLAKE3	$\mathtt{H}(x) \sim U(0, 2^{k})$	$\mathtt{H}(\mathtt{maul}(x))$	Yes	0

Outline

RQ1. Experimentation Details

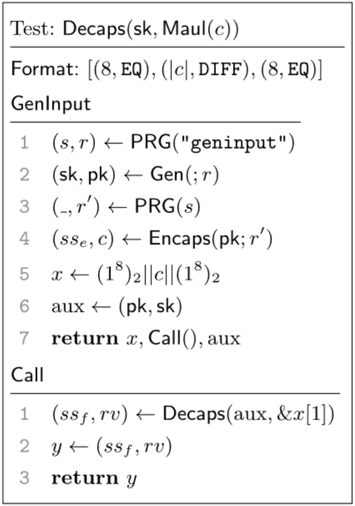
- Metamorphic Tests
- Statistical Tests

Kyber KEM - Test 2

```
Test: $\mathsf{Encaps}(\mathsf{Maul}(\mathsf{pk}); r)$
Format: $[(8, \texttt{EQ}), (|\mathsf{pk}|, \texttt{DIFF}), (8, \texttt{EQ})]$
GenInput
    $(\_, r) \leftarrow \mathsf{PRG}(\texttt{"geninput"})$
    $(\mathsf{pk}, \mathsf{sk}, rv) \leftarrow \mathsf{Gen}(; r)$
    $x \leftarrow (1^8)_2 || \mathsf{pk} || (1^8)_2$
    $\mathrm{aux} \leftarrow \mathsf{sk}$
    return $x, \mathsf{Call}(), \mathrm{aux}$
Call
    $(\_, r) \leftarrow \mathsf{PRG}(\texttt{"call"})$
    $(ss, c, rv) \leftarrow \mathsf{Encaps}(\&x[1]; r)$
    $y \leftarrow (ss || c, rv)$
    return $y$
```

```
Test: $\mathsf{Decaps}(\mathsf{sk}, \mathsf{Encaps}(\mathsf{Maul}(\mathsf{pk}); r))$
Format: $[(8, \texttt{EQ}), (|\mathsf{pk}|, \texttt{DIFF}), (8, \texttt{EQ})]$
GenInput
    $(\_, r) \leftarrow \mathsf{PRG}(\texttt{"geninput"})$
    $(\mathsf{pk}, \mathsf{sk}, rv) \leftarrow \mathsf{Gen}(; r)$
    $x \leftarrow (1^8)_2 || \mathsf{pk} || (1^8)_2$
    $\mathrm{aux} \leftarrow \mathsf{sk}$
    return $x, \mathsf{Call}(), \mathrm{aux}$
Call
    $(\_, r) \leftarrow \mathsf{PRG}(\texttt{"call"})$
    $(ss_e, c) \leftarrow \mathsf{Encaps}(\&x[1]; r)$
    $(ss_f, rv) \leftarrow \mathsf{Decaps}(c, \mathrm{aux})$
    $eq \leftarrow [\![ss_e = ss_f]\!]$
    $y \leftarrow (eq, rv)$
    return $y$
```

Kyber KEM - Test 3



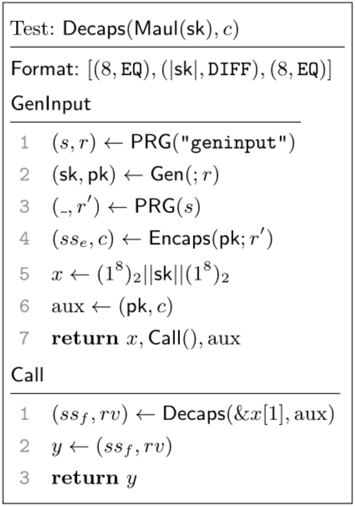
```
Test: Decaps(sk, Maul($c$))

Format: $[(8, \texttt{EQ}), (|c|, \texttt{DIFF}), (8, \texttt{EQ})]$

GenInput
    $(s, r) \leftarrow \mathsf{PRG}(\texttt{"geninput"})$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(; r)$
    $(\_, r') \leftarrow \mathsf{PRG}(s)$
    $(ss_e, c) \leftarrow \mathsf{Encaps}(\mathsf{pk}; r')$
    $x \leftarrow (1^8)_2||c||(1^8)_2$
    $\mathrm{aux} \leftarrow (\mathsf{pk}, \mathsf{sk})$
    return $x$, Call(), aux

Call
    $(ss_f, rv) \leftarrow \mathsf{Decaps}(\mathrm{aux}, \&x[1])$
    $y \leftarrow (ss_f, rv)$
    return $y$
```

Kyber KEM - Test 4 (1)



Test: Decaps(Maul(sk), c)

Format: $[(8, \texttt{EQ}), (|\mathsf{sk}|, \texttt{DIFF}), (8, \texttt{EQ})]$

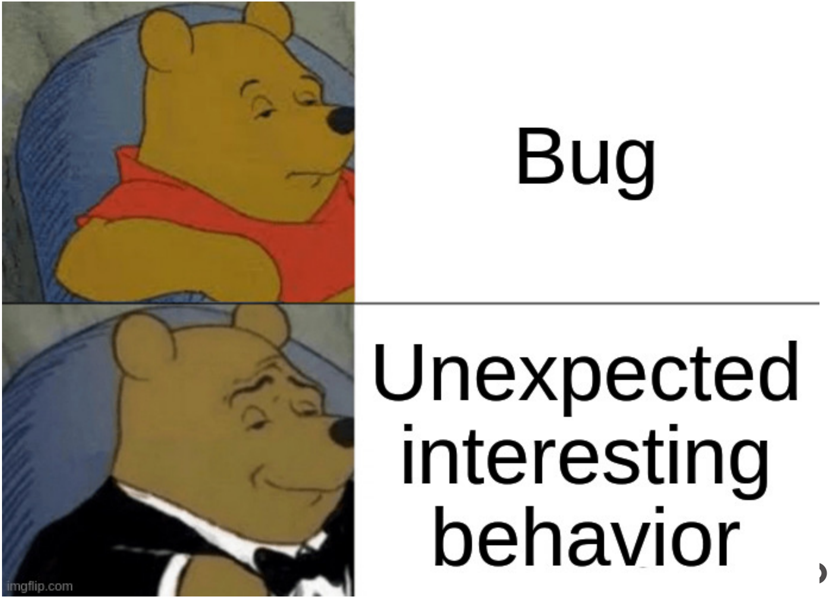
GenInput

```
$(s, r) \leftarrow$ PRG("geninput")
$(\mathsf{sk}, \mathsf{pk}) \leftarrow$ Gen$(; r)$
$(\_, r') \leftarrow$ PRG$(s)$
$(ss_e, c) \leftarrow$ Encaps$(\mathsf{pk}; r')$
$x \leftarrow (1^8)_2 || \mathsf{sk} || (1^8)_2$
$\mathrm{aux} \leftarrow (\mathsf{pk}, c)$
return $x$, Call(), $\mathrm{aux}$
```

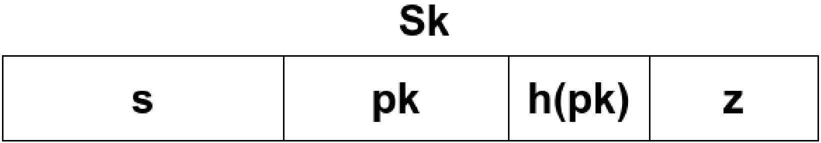
Call

```
$(ss_f, rv) \leftarrow$ Decaps$(\&x[1], \mathrm{aux})$
$y \leftarrow (ss_f, rv)$
return $y$
```

2 interpretations ...



Kyber KEM - Test 4 (2)



Sk

s	pk	h(pk)	z

Figure: Kyber KEM secret key layout

Testing SHA-1,2,3

```
Test: $\mathsf{Hash}(\mathsf{Maul}(x))$

Format: $[(8, \texttt{EQ}), (\ell, \texttt{DIFF}), (8, \texttt{EQ})]$

GenInput
    $x \leftarrow (1^8)_2||(0^\ell)_2||(1^8)_2$
    $\mathrm{aux} \leftarrow \perp$
    $\textbf{return } x, \mathsf{Call}(), \mathrm{aux}$

Call
    $y \leftarrow \mathsf{Hash}(\&x[1])$
    $\textbf{return } y$
```

Outline

RQ1. Experimentation Details

- Metamorphic Tests
- Statistical Tests

statistical tests

- Statistical tests check the quality of pseudo random bit generators.

statistical tests

- Statistical tests check the quality of pseudo random bit generators.
- They ensure bit distribution “looks like random"

statistical tests

- Statistical tests check the quality of pseudo random bit generators.
- They ensure bit distribution “looks like random"
- It’s like ensuring that a dice isn’t loaded by throwing it multiple times

statistical tests

- Statistical tests check the quality of pseudo random bit generators.
- They ensure bit distribution “looks like random"
- It’s like ensuring that a dice isn’t loaded by throwing it multiple times
- We used TestU01 test suite on outputs from different HMAC and hash compression algorithms

Outline

RQ2. Writing a Rust library

- Introduction to Rust
- Rust Crypto
- Our library : `metamorphic-testing-rs`

Rust

What is Rust ?

Rust

What is Rust ?

- Compiled programming language

Rust

What is Rust ?

- Compiled programming language
- Strongly typed

Rust

What is Rust ?

- Compiled programming language
- Strongly typed
- Emphasis on memory safety

Rust

What is Rust ?

- Compiled programming language
- Strongly typed
- Emphasis on memory safety
- Try to avoid runtime errors

Rust

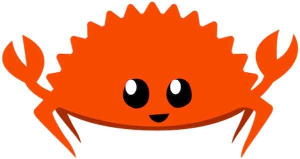
What is Rust ?

- Compiled programming language
- Strongly typed
- Emphasis on memory safety
- Try to avoid runtime errors
- Many critical use-cases (Network stack, Kernel, . . .)

Rust

What is Rust ?

- Compiled programming language
- Strongly typed
- Emphasis on memory safety
- Try to avoid runtime errors
- Many critical use-cases (Network stack, Kernel, . . .)
- Best programming language



Outline

RQ2. Writing a Rust library

- Introduction to Rust
- Rust Crypto
- Our library : `metamorphic-testing-rs`

Rust crypto

Project which aims to centralize and maintain many cryptographic implementations in Rust

Rust crypto

Project which aims to centralize and maintain many cryptographic implementations in Rust

- Gather many cryptographic primitives
- Standard shared APIs to use families of primitives
- Good documentation (known attacks, recommended key sizes and primitives, ...)

Rust crypto

Project which aims to centralize and maintain many cryptographic implementations in Rust

- Gather many cryptographic primitives
- Standard shared APIs to use families of primitives
- Good documentation (known attacks, recommended key sizes and primitives, ...)

Biggest cryptographic library but others do exist (Ring...)

How to test crypto function with Rust ?

```
use sha3::{Digest, Sha3_256};

let mut hasher = Sha3_256::new();
hasher.update(b"abc");
let hash = hasher.finalize();
```

Figure: Rust code to use SHA3 256

How to test crypto function with Rust ?

How to test crypto function with Rust ?

A metamorphic test is a set of five functions : `GenInput`, `GenState`, `Call`, `Maul`, `Check`

How to test crypto function with Rust ?

A metamorphic test is a set of five functions : `GenInput`, `GenState`, `Call`, `Maul`, `Check`

```
$\sigma \leftarrow \texttt{GenState}()$
$x \leftarrow \texttt{GenInput}(n)$
$y \leftarrow \texttt{Call}(\sigma, x)$
for $i$ in $1, \ldots,$ runs do
    $\sigma', x' \leftarrow \texttt{Maul}(\sigma, x, i)$
    $y' \leftarrow \texttt{Call}(\sigma', x')$
    $\texttt{Check}(y, y')$
end for
```

Figure: Fenzi inspired test framework

Applying our framework to SHA in Rust

How to apply our Bit Inclusion test to SHA3 256 ?

Applying our framework to SHA in Rust

How to apply our Bit Inclusion test to SHA3 256 ?

```
fn gen_state() -> Sha3_256 {
    return Sha3_256::new();
}

fn gen_input(n: usize) -> Vec<u8> {
    return Rand::randbytes(n).to_vec();
}

fn call(state: Sha3_256, input: Vec<u8>) -> Hash{
    state.update(input);
    return state.finalize();
}

fn maul(state: Sha3_256, input: Vec<u8>, i: usize) -> (Sha3_256, Vec<u8>) {
    let output = input;
    flip_bit_at_index(&mut output, i);
    return (state, output);
}

fn check(ref_output: Vec<u8>, output: Vec<u8>) -> bool {
    return ref_output != output;
}
```

Outline

RQ2. Writing a Rust library

- Introduction to Rust
- Rust Crypto
- Our library : `metamorphic-testing-rs`

Our library

Rust library[2] allowing to create and run metamorphic tests

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries
- Set of traits to build metamorphic tests

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries
- Set of traits to build metamorphic tests
- Mainly used for hash functions (47 hash functions tested) but can easily be extended.

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries
- Set of traits to build metamorphic tests
- Mainly used for hash functions (47 hash functions tested) but can easily be extended.
- Only two lines of code to add a new hash implementation from Rust Crypto

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries
- Set of traits to build metamorphic tests
- Mainly used for hash functions (47 hash functions tested) but can easily be extended.
- Only two lines of code to add a new hash implementation from Rust Crypto
- Run tests in parallel to improve speed

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Our library

Rust library[2] allowing to create and run metamorphic tests

- Set of traits (interfaces) to plug cryptographic libraries
- Set of traits to build metamorphic tests
- Mainly used for hash functions (47 hash functions tested) but can easily be extended.
- Only two lines of code to add a new hash implementation from Rust Crypto
- Run tests in parallel to improve speed
- Free and open source

[2] https://github.com/thomasarmel/metamorphic-testing-rs

Outline

Conclusions

Key Takeaways

Key Takeways

1. Metamorphic testing is useful when no oracle is available

Key Takeways

1. Metamorphic testing is useful when no oracle is available
2. ML-KEM Kyber implementation is not compliant with NIST specificities

Key Takeways

1. Metamorphic testing is useful when no oracle is available
2. ML-KEM Kyber implementation is not compliant with NIST specificities
3. Difficult to build highly generic library for testing

Key Takeways

1. Metamorphic testing is useful when no oracle is available
2. ML-KEM Kyber implementation is not compliant with NIST specificities
3. Difficult to build highly generic library for testing
4. Rust Crypto seems pretty safe at first glance

Key Takeways

1. Metamorphic testing is useful when no oracle is available
2. ML-KEM Kyber implementation is not compliant with NIST specificities
3. Difficult to build highly generic library for testing
4. Rust Crypto seems pretty safe at first glance
5. Future works : test more primitives (NTLM ;))

Key Takeways

1. Metamorphic testing is useful when no oracle is available
2. ML-KEM Kyber implementation is not compliant with NIST specificities
3. Difficult to build highly generic library for testing
4. Rust Crypto seems pretty safe at first glance
5. Future works : test more primitives (NTLM ;))

Thank you !