Cryptography
Research Center

CryptographicEstimators

A Software Library for Cryptographic Hardness Estimation

Andre Esser, Javier Verbel, Floyd Zweydinger and Emanuele Bellini

@NIST PQC Seminar, Nov. 2023

Technology
Innovation
Institute

Contents

Introduction

Cryptographic Hardness Estimation

Cryptographic Hardness Estimation

Estimation of required time to solve a (cryptographic) problem

- Security guarantees
- Parameter selection
  - Example: RSA keysize recommendations
- Estimates change over time: adaptive process

The Case of Classical Cryptography

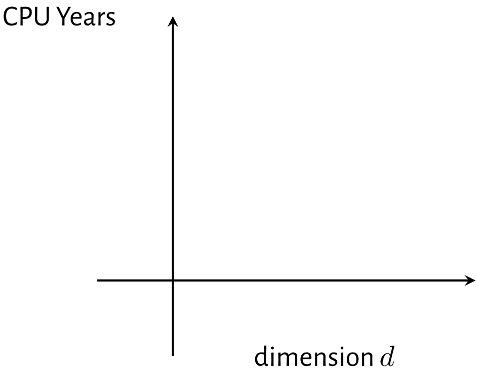
- Established methodology

The Case of Classical Cryptography

- Established methodology
    1. Theory
    2. Experiments
    3. Extrapolate

The Case of Classical Cryptography

runtime formula T_d

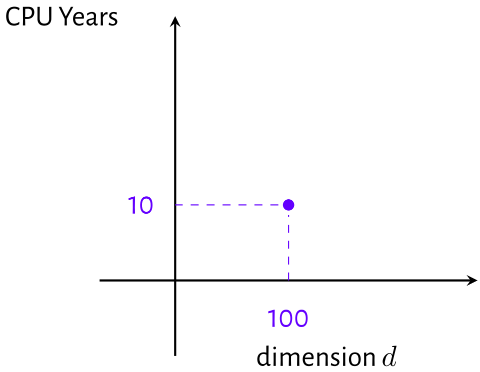


- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

The Case of Classical Cryptography

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

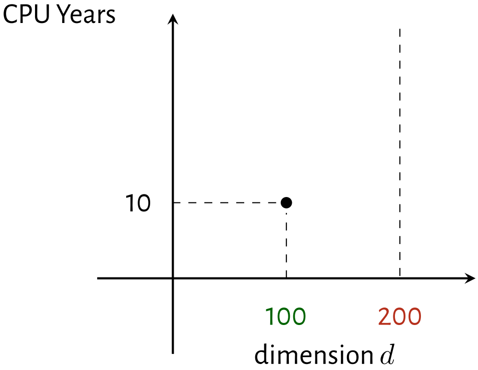
runtime formula T_d



The Case of Classical Cryptography

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

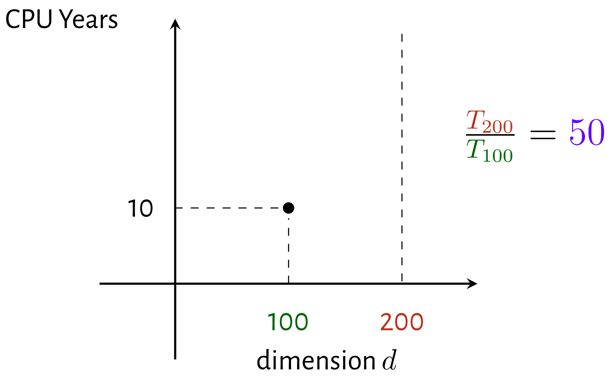
runtime formula T_d



The Case of Classical Cryptography

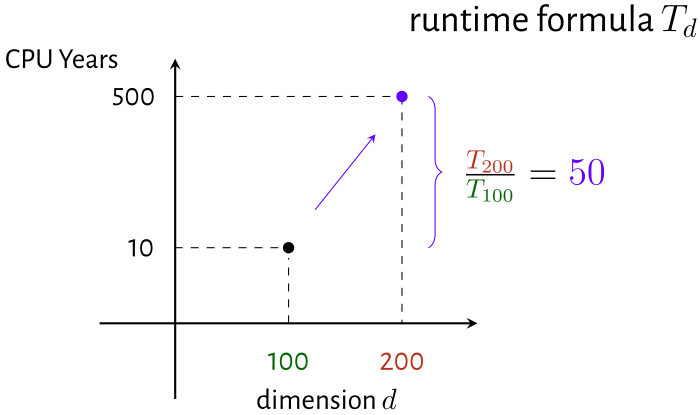
- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

runtime formula T_d



The Case of Classical Cryptography

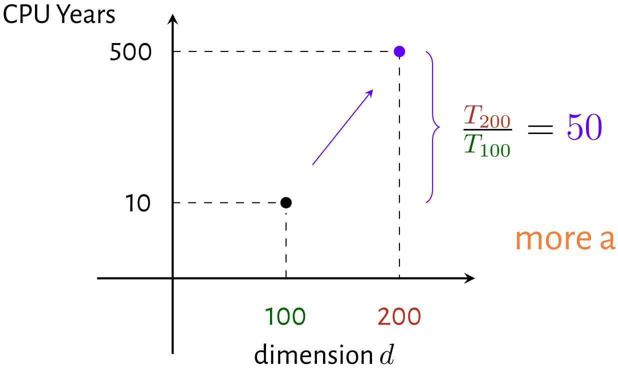
- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate



The Case of Classical Cryptography

runtime formula T_d

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

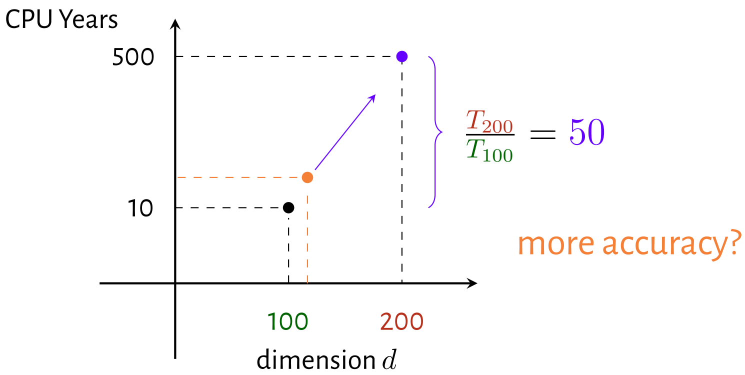


more accuracy?

The Case of Classical Cryptography

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

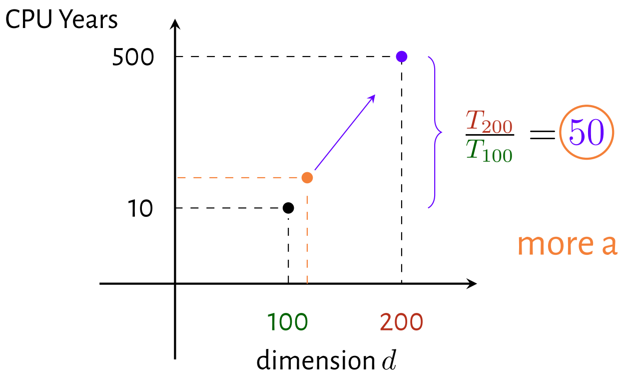
runtime formula T_d



The Case of Classical Cryptography

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate

runtime formula T_d

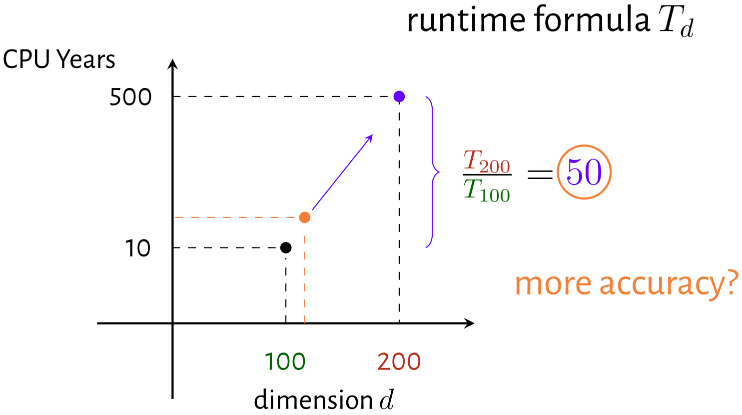


more accuracy?

The Case of Classical Cryptography

Assumption: Scalability

- Established methodology
  1. Theory
  2. Experiments
  3. Extrapolate



The Case of PQC

- Difficult scalability (memory)

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

Time to solve a problem = Time of fastest known algorithm

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

Time to solve a problem = Time of fastest known algorithm

- Hardness depends on best known algorithms

- Requires estimation of time of all known algorithms

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

Time to solve a problem = Time of fastest known algorithm

- Hardness depends on best known algorithms

- Requires estimation of time of all known algorithms

- Main Challenges

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

Time to solve a problem = Time of fastest known algorithm

- Hardness depends on best known algorithms

- Requires estimation of time of all known algorithms

- Main Challenges
  - Consensus

The Case of PQC

- Difficult scalability (memory) $\Rightarrow$ estimation methodology

Time to solve a problem = Time of fastest known algorithm

- Hardness depends on best known algorithms

- Requires estimation of time of all known algorithms

- Main Challenges
  - Consensus
  - Accessibility

CryptographicEstimators

CryptographicEstimators

Python / Sage library for estimations of cryptographic problems

CryptographicEstimators

Python / Sage library for estimations of cryptographic problems

- Main goals
  - State-of-the-art estimations
  - Centralization of estimation efforts
  - Community-driven open-source project
  - Easy accessibility

CryptographicEstimators

Python / Sage library for estimations of cryptographic problems

- Main goals
  - State-of-the-art estimations
  - Centralization of estimation efforts
  - Community-driven open-source project
  - Easy accessibility
- Current State: 6 Estimators, 32 Algorithms
  - Multivariate Quadratic (MQ)
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq)
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE)
  - Permuted Kernel (PK)

Theoretical Considerations

What to Estimate?

Estimate required time (and memory) of algorithm $\mathcal{A}$ to solve problem $\mathcal{P}$

What to Estimate?

Estimate required time (and memory) of algorithm $\mathcal{A}$ to solve problem $\mathcal{P}$

- Time : Measured in basic *operations* `op`
- Memory: Measured in basic *elements* `el`

What to Estimate?

Estimate required time (and memory) of algorithm $\mathcal{A}$ to solve problem $\mathcal{P}$

- Time : Measured in basic *operations* $\texttt{op}$
- Memory: Measured in basic *elements* $\texttt{el}$

- Basic units depend on problem $\mathcal{P}$
  - Example MQ-Problem : $\texttt{op}:$ $\mathbb{F}_q$-multiplication and $\texttt{el}:$ $\mathbb{F}_q$-element
  - Example binary SD-Problem: $\texttt{op}:$ $\mathbb{F}_2^n$-vector addition and $\texttt{el}:$ $\mathbb{F}_2^n$-vector

What to Estimate?

Estimate required time (and memory) of algorithm $\mathcal{A}$ to solve problem $\mathcal{P}$

- Time : Measured in basic *operations* $\texttt{op}$
- Memory: Measured in basic *elements* $\texttt{el}$

- Basic units depend on problem $\mathcal{P}$
  - Example MQ-Problem : $\texttt{op}$: $\mathbb{F}_q$-multiplication and $\texttt{el}$: $\mathbb{F}_q$-element
  - Example binary SD-Problem: $\texttt{op}$: $\mathbb{F}_2^n$-vector addition and $\texttt{el}$: $\mathbb{F}_2^n$-vector

- Problem defines $\texttt{op}$ / $\texttt{el}$ to bit (operation) conversion

Memory Access Costs

Memory Access Costs

Computations are performed in the RAM model

Memory Access Costs

Computations are performed in the RAM model

- Accessing 1 bit equals 1 bit operation

Memory Access Costs

Computations are performed in the RAM model

- Accessing 1 bit equals 1 bit operation
- Embedding higher cost
  - Accessing 1 bit in memory of size M takes $f(M) = \sqrt{M}, \sqrt[3]{M}, \ldots$ bit operations

Memory Access Costs

Computations are performed in the RAM model

- Accessing 1 bit equals 1 bit operation
- Embedding higher cost
  - Accessing 1 bit in memory of size M takes $f(M) = \sqrt{M}, \sqrt[3]{M}, \ldots$ bit operations
- Upper bound on memory access: $T \cdot f(M)$

Memory Access Costs

Computations are performed in the RAM model

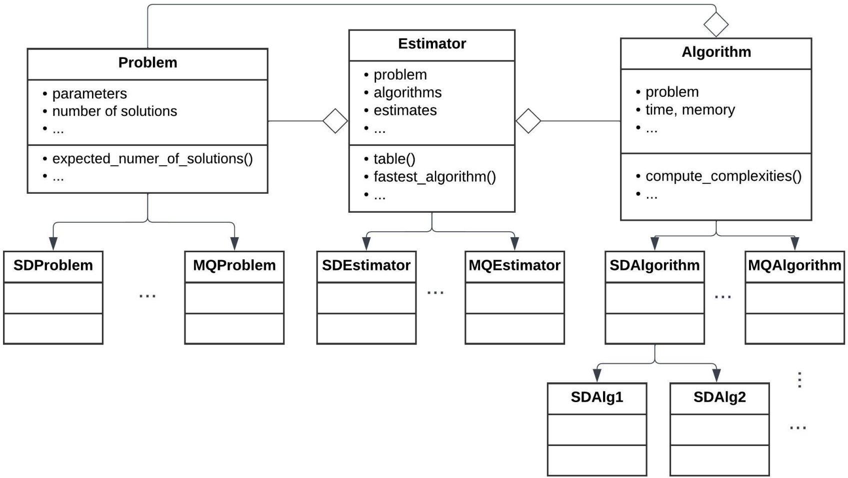
- Accessing 1 bit equals 1 bit operation
- Embedding higher cost
  - Accessing 1 bit in memory of size M takes $f(M) = \sqrt{M}, \sqrt[3]{M}, \ldots$ bit operations
- Upper bound on memory access: $T \cdot f(M)$
  - Real cost C of the full algorithm : $T \leq C \leq T \cdot f(M)$

Technical Design

Class Design

Class Design

CryptographicEstimators: An object-oriented Python library



Usage

Usage

- From command line:

- From web application [2]:

[1]Docker: `https://www.docker.com`
[2]Webapp: `https://estimators.crypto.tii.ae`

Usage

- From command line:
  - Install –> sage –>

```
from cryptographic_estimators.SDEstimator import SDEstimator
A = SDEstimator(n=500, k=250, w=50)
A.table()
```

- From web application [2]:

[1]Docker: https://www.docker.com

[2]Webapp: https://estimators.crypto.tii.ae

Usage

- From command line:
  - Install –> sage –>
- From web application [2]:

	estimate	
algorithm	time	memory
BallCollision	64.8	22.9
BJMMdw	64.0	25.2
BJMMpdw	63.9	28.0
BJMM	63.6	29.4
BJMM_plus	63.5	26.5
BothMay	63.2	25.2
Dumer	63.9	28.5
MayOzerov	62.7	34.0
Prange	77.7	17.3
Stern	63.8	22.9

[1] Docker: `https://www.docker.com`
[2] Webapp: `https://estimators.crypto.tii.ae`

Usage

- From command line:
  - Install –> `sage` –>
  - Docker[1] –> `make docker-run` –> same as local
- From web application [2]:

	estimate	
algorithm	time	memory
BallCollision	64.8	22.9
BJMMdw	64.0	25.2
BJMMpdw	63.9	28.0
BJMM	63.6	29.4
BJMM_plus	63.5	26.5
BothMay	63.2	25.2
Dumer	63.9	28.5
MayOzerov	62.7	34.0
Prange	77.7	17.3
Stern	63.8	22.9

[1] Docker: `https://www.docker.com`
[2] Webapp: `https://estimators.crypto.tii.ae`

Usage

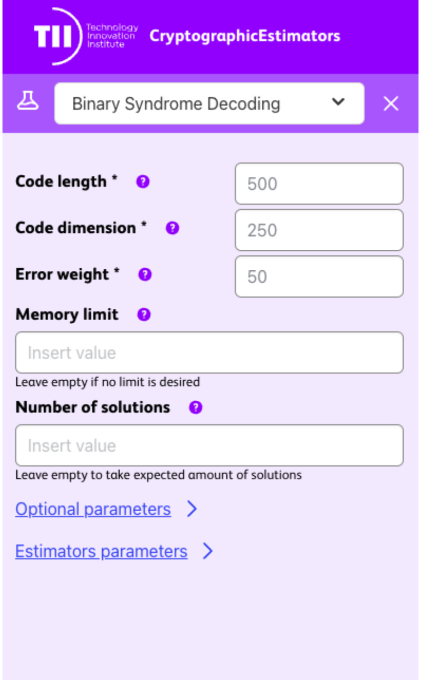
- From command line:
  - Install –> `sage` –>
  - Docker[1] –> `make docker-run` –> same as local
- From web application [2]:

	estimate	
algorithm	time	memory
BallCollision	64.8	22.9
BJMMdw	64.0	25.2
BJMMpdw	63.9	28.0
BJMM	63.6	29.4
BJMM_plus	63.5	26.5
BothMay	63.2	25.2
Dumer	63.9	28.5
MayOzerov	62.7	34.0
Prange	77.7	17.3
Stern	63.8	22.9

[1] Docker: `https://www.docker.com`
[2] Webapp: `https://estimators.crypto.tii.ae`

Usage

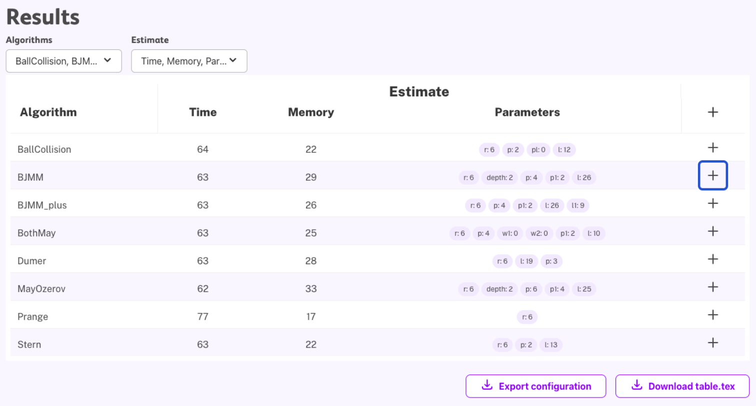
- From command line:
  - Install –> sage –>
  - Docker[1] –> `make docker-run` –> same as local
- From web application[2]:



[1]Docker: `https://www.docker.com`
[2]Webapp: `https://estimators.crypto.tii.ae`

Usage

- From command line:
  - Install –> sage –>
  - Docker[1] –> `make docker-run`
- From web application[2]:



Algorithm	Time	Memory	Parameters	+
BallCollision	64	22	r: 6, p: 2, pl: 0, l: 12	+
BJMM	63	29	r: 6, depth: 2, p: 4, p1: 2, l: 26	+
BJMM_plus	63	26	r: 6, p: 4, p1: 2, l: 26, l1: 9	+
BothMay	63	25	r: 6, p: 4, w1: 0, w2: 0, p1: 2, l: 10	+
Dumer	63	28	r: 6, l: 19, p: 3	+
MayOzerov	62	33	r: 6, depth: 2, p: 6, p1: 4, l: 25	+
Prange	77	17	r: 6	+
Stern	63	22	r: 6, p: 2, l: 13	+

[1]Docker: `https://www.docker.com`
[2]Webapp: `https://estimators.crypto.tii.ae`

More Functionalities

- Available algorithms:

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

  complexity of crossbred –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity()
70.8336959616846
```

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:
  optimal parameters –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.optimal_parameters()
{'D': 9, 'd': 1, 'k': 11}
```

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

complex. for $(D, d, k) = (6, 1, 3)$ –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity(k=3, D=6, d=1)
98.20496250072115
```

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

complex. for $(D, d, k) = (6, 1, 3)$ –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity(k=3, D=6, d=1)
98.20496250072115
```

- More functionalities:

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

complex. for $(D, d, k) = (6, 1, 3)$ –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity(k=3, D=6, d=1)
98.20496250072115
```

- More functionalities:
  - Time complexity –> given as basic operations or bit operations.

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

complex. for $(D, d, k) = (6, 1, 3)$ –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity(k=3, D=6, d=1)
98.20496250072115
```

- More functionalities:
  - Time complexity –> given as basic operations or bit operations.
  - Optimization under constraints, e.g:
    1. memory bounds.
    2. restricted parameters ranges.

More Functionalities

- Available algorithms:

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.algorithm_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

complex. for $(D, d, k) = (6, 1, 3)$ –>

```
sage: from cryptographic_estimators.MQEstimator import MQEstimator
sage: E = MQEstimator(n=24, m=24, q=16)
sage: E.crossbred.time_complexity(k=3, D=6, d=1)
98.20496250072115
```

- More functionalities:
  - Time complexity –> given as basic operations or bit operations.
  - Optimization under constraints, e.g:
    1. memory bounds.
    2. restricted parameters ranges.

A full user guide is available.

Contributing

Contributing

- **Where?** –> Public repository `CryptographicEstimators` [3].

[3] `https://github.com/Crypto-TII/CryptographicEstimators`.

Contributing

- **Where?** –> Public repository CryptographicEstimators [3].
- **Submitting code?** –> Write access by an email to cryptographic_estimators@tii.ae.

[3] https://github.com/Crypto-TII/CryptographicEstimators.

Contributing

- **Where?** –> Public repository `CryptographicEstimators`[3].
- **Submitting code?** –> Write access by an email to `cryptographic_estimators@tii.ae`.
- **Guidelines?** –> Check CONTRIBUTING.md in the repository.

[3] `https://github.com/Crypto-TII/CryptographicEstimators`.

Contributing

- **Where?** –> Public repository `CryptographicEstimators` [3].
- **Submitting code?** –> Write access by an email to `cryptographic_estimators@tii.ae`.
- **Guidelines?** –> Check CONTRIBUTING.md in the repository.
- **Don't want to code?** –>

[3] `https://github.com/Crypto-TII/CryptographicEstimators`.

Contributing

- **Where?** –> Public repository `CryptographicEstimators` [3].
- **Submitting code?** –> Write access by an email to `cryptographic_estimators@tii.ae`.
- **Guidelines?** –> Check CONTRIBUTING.md in the repository.
- **Don't want to code?** –>
  1. Check the code and raise issues.
  2. Participate in the discussion (within the repository).

[3] `https://github.com/Crypto-TII/CryptographicEstimators`.

NIST PQC Signatures

Coverage

- Current included estimators:

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ)

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq)

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE) —-> $\mathbf{G}' = \mathbf{SGQ}$, with $\mathbf{Q}$ a monomial matrix.

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE) —-> $\mathbf{G}' = \mathbf{SGQ}$, with $\mathbf{Q}$ a monomial matrix.

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE) —-> $\mathbf{G}' = \mathbf{SGQ}$, with $\mathbf{Q}$ a monomial matrix.
  - Permuted Kernel (PK)

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE) —-> $\mathbf{G}' = \mathbf{SGQ}$, with $\mathbf{Q}$ a monomial matrix.
  - Permuted Kernel (PK) —-> $\mathbf{0} = \mathbf{H} \cdot \pi(\mathbf{x})$, with π a permuation.

Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ) —-> $\mathbf{y} = \mathcal{P}(\mathbf{x})$, with $\mathcal{P}$ a quadratic map.
  - Binary Syndrome Decoding (SD)
  - Syndrome Decoding over $\mathbb{F}_q$ (SDFq) —-> $\mathbf{s} = \mathbf{H} \cdot \mathbf{e}$, with $wt(\mathbf{e}) \leq \omega$.
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE) —-> $\mathbf{G}' = \mathbf{SGQ}$, with $\mathbf{Q}$ a monomial matrix.
  - Permuted Kernel (PK) —-> $\mathbf{0} = \mathbf{H} \cdot \pi(\mathbf{x})$, with π a permuation.

- Best known attacks of 8 / 30 (remaining) submissions fall into this scope

Estimation of NIST Candidates

- Estimates for NIST Category I parameter sets

Scheme	Hardness Assumption	Est. Time	Est. Memory
SDitH	SDFq	147.0	26.9
LESS	LE	136.6	39.0
PERK	PK	155.5	154.4
MQOM	MQ	142.8	51.8
TUOV / UOV	MQ	144.5	59.6
VOX	MQ	153.0	59.8
PROV	MQ	150.1	62.3

Future Developments

Future Developments

- We plan to maintain the library, and actively develop it.

Future Developments

- We plan to maintain the library, and actively develop it.

- Features in development
  - Scheme vs. Problem estimators
  - Advanced memory access

Future Developments

- We plan to maintain the library, and actively develop it.
- Features in development
  - Scheme vs. Problem estimators
  - Advanced memory access
- Estimators in development

Future Developments

- We plan to maintain the library, and actively develop it.

- Features in development
  - Scheme vs. Problem estimators
  - Advanced memory access

- Estimators in development
  - Schemes: UOV –> National University of Colombia and TII.

Future Developments

- We plan to maintain the library, and actively develop it.

- Features in development
  - Scheme vs. Problem estimators
  - Advanced memory access

- Estimators in development
  - Schemes: UOV –> National University of Colombia and TII.
  - Problems:
    - MinRank –> NIST, TII and University of Limoges
    - Rank Syndrome Decoding –> NIST, TII and University of Limoges.
    - Rank Support Learning –> NIST, TII and University of Limoges.
    - Regular Syndrome Decoding –> Marche Polytechnic University, TII, and potential other collaborators.

Future Developments

- We plan to maintain the library, and actively develop it.
- Features in development
  - Scheme vs. Problem estimators
  - Advanced memory access
- Estimators in development
  - Schemes: UOV –> National University of Colombia and TII.
  - Problems:
    - MinRank –> NIST, TII and University of Limoges
    - Rank Syndrome Decoding –> NIST, TII and University of Limoges.
    - Rank Support Learning –> NIST, TII and University of Limoges.
    - Regular Syndrome Decoding –> Marche Polytechnic University, TII, and potential other collaborators.

Thank you!

TII Technology Innovation Institute

tii.ae