CryptographicEstimators
A Software Library for Cryptographic Hardness Estimation

Andre Esser, Javier Verbel, Floyd Zweydinger and Emanuele Bellini

@NIST PQC Seminar, Nov. 2023
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

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

runtime formula $T_d$

CPU Years

dimension $d$
The Case of Classical Cryptography

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

The runtime formula is $T_d$. The graph shows the relationship between CPU Years and dimension $d$. At 100 CPU Years, the dimension $d$ is extrapolated to be 10.
The Case of Classical Cryptography

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

runtime formula $T_d$

CPU Years

100 200

dimension $d$

10
The Case of Classical Cryptography

Established methodology

1. Theory
2. Experiments
3. Extrapolate

runtime formula $T_d$

$$\frac{T_{200}}{T_{100}} = 50$$
The Case of Classical Cryptography

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

![Graph with runtime formula](image)

runtime formula $T_d$

- CPU Years
- Dimension $d$

Graph with points labeled 10, 100, 500 and 200, connected by lines to illustrate the runtime formula $\frac{T_{200}}{T_{100}} = 50$.
The Case of Classical Cryptography

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

CPU Years

runtime formula $T_d$

$\frac{T_{200}}{T_{100}} = 50$

more accuracy?
The Case of Classical Cryptography

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

- Runtime formula $T_d$

```
10 100 200

500

10

T_{200} / T_{100} = 50

more accuracy?
```
The Case of Classical Cryptography

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

runtime formula $T_d$

$$\frac{T_{200}}{T_{100}} = 50$$

more accuracy?
The Case of Classical Cryptography

Established methodology

1. Theory
2. Experiments
3. Extrapolate

Assumption: Scalability

runtime formula $T_d$

more accuracy?

$\frac{T_{200}}{T_{100}} = 50$
The Case of PQC

- Difficult scalability (memory)
The Case of PQC

- Difficult scalability (memory) ⇒ estimation methodology
The Case of PQC

- Difficult scalability (memory) ⇒ estimation methodology

Time to solve a problem = Time of fastest known algorithm
The Case of PQC

- Difficult scalability (memory) ⇒ estimation methodology

\[
\text{Time to solve a problem} = \text{Time of fastest known algorithm}
\]

- 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) ⇒ 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) ⇒ 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

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 \( \mathbb{F}_q \) (SDF\( q \))
- Permutation Equivalence (PE)
- Linear Equivalence (LE)
- Permuted Kernel (PK)
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)
- Permutated Kernel (PK)
Theoretical Considerations
What to Estimate?

Estimate required time (and memory) of algorithm $A$ to solve problem $P$.
What to Estimate?

Estimate required time (and memory) of algorithm $A$ to solve problem $P$

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

Example MQ-Problem:
- $op$: $F_q^*$-multiplication
- $el$: $F_q^*$-element

Example binary SD-Problem:
- $op$: $F_{n^2}$-vector addition
- $el$: $F_{n^2}$-vector

Problem defines $op/\ el$ to bit (operation) conversion.
What to Estimate?

Estimate required time (and memory) of algorithm $A$ to solve problem $P$.

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

- Basic units depend on problem $P$.
  - Example MQ-Problem: $\text{op}: \mathbb{F}_q$-multiplication and $\text{el}: \mathbb{F}_q$-element.
  - Example binary SD-Problem: $\text{op}: \mathbb{F}_2^n$-vector addition and $\text{el}: \mathbb{F}_2^n$-vector.
What to Estimate?

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

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

Basic units depend on problem $\mathcal{P}$

- Example MQ-Problem: $\text{op}: \mathbb{F}_q$-multiplication and $\text{el}: \mathbb{F}_q$-element
- Example binary SD-Problem: $\text{op}: \mathbb{F}_2^n$-vector addition and $\text{el}: \mathbb{F}_2^n$-vector

- Problem defines $\text{op} / \text{el}$ to bit (operation) conversion
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 $f(M) = \sqrt[6]{M}$, $\sqrt[3]{M}$, ... bit operations. Upper bound on memory access: $T \cdot f(M)$. Real cost $C$ of the full algorithm: $T \leq C \leq T \cdot f(M)$.
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: $T \cdot f(M)$
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 $f(M) = \sqrt{M}, \frac{3}{\sqrt[3]{M}}, \ldots$ 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

- **Problem**
  - parameters
  - number of solutions
  - ...
  - expected_numer_of_solutions()
  - ...

- **Estimator**
  - problem
  - algorithms
  - estimates
  - ...
  - table()
  - fastest_algorithm()
  - ...

- **Algorithm**
  - problem
  - time, memory
  - ...
  - compute_complexities()
  - ...

SDProblem ...
MQProblem ...
SDEstimator ...
MQEstimator ...
SDAlgorithm ...
MQAlgorithm ...
SDAlg1 ...
SDAlg2 ...
Usage
Usage

- From command line:

- From web application\(^2\):

---

\(^1\)Docker: https://www.docker.com
\(^2\)Webapp: https://estimators.cryptotii.ae
Usage

- From command line:
  - Install → sage →

- 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:\(^2\):

---

1Docker: https://www.docker.com
2Webapp: 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.cryptotii.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.cryptotii.ae
Usage

- From command line:
  - Install → sage →
  - Docker → make docker-run → same as local

- From web application:

---

1Docker: https://www.docker.com
2Webapp: https://estimators.crypto.tii.ae
Usage

- From command line:
  - Install -> sage ->
- Docker¹ -> make docker-run
- From web application²:

¹Docker: https://www.docker.com
²Webapp: https://estimators.crypto.tii.ae
More Functionalities

- Available algorithms:

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

- Optimization under constraints, e.g:
  1. memory bounds.
  2. restricted parameters ranges.

- Complexity of crossbred –> optimal parameters –> complex. for \((D, d, k) = (6, 1, 3)\) –> A full user guide is available.
More Functionalities

- Available algorithms:

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

- Available algorithms:

- Access single algorithms:

```python
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:**
  - Access single algorithms: complexity of crossbred →

```python
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']
```

```python
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:

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

```python
sage: E.algorithms_names()
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms: optimal parameters →

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

```python
sage: E.crossbred_optimal_parameters()
{'D': 9, 'd': 1, 'k': 11}
```
More Functionalities

- Available algorithms:

- Access single algorithms:
  complex. for \((D, d, k) = (6, 1, 3)\) →

```python
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']
```

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

- Available algorithms:

```python
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:

```python
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)
```

- More functionalities:
More Functionalities

- Available algorithms:

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

```
['BooleanSolveFXL',
 'Crossbred',
 'ExhaustiveSearch',
 'F5',
 'HybridF5',
 'Lokshtanov']
```

- Access single algorithms:

```python
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.

Complexity of crossbred -> optimal parameters -> complex. for \((D, d, k) = (6, 1, 3)\)
More Functionalities

■ Available algorithms:

- Access single algorithms:
  complex. for \((D, d, k) = (6, 1, 3)\) →

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

■ Available algorithms:

```python
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:

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

■ More functionalites:

■ 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 CryptographicEstimators³.

Contributing

- **Where?** → Public repository CryptographicEstimators³.

- **Submitting code?** → Write access by an email to cryptographic_estimators@tii.ae.

Contributing

- **Where?** → Public repository CryptographicEstimators³.

- **Submitting code?** → Write access by an email to cryptographic_estimators@tii.ae.

- **Guidelines?** → Check CONTRIBUTING.md in the repository.

Contributing

- **Where?** → Public repository CryptographicEstimators³.

- **Submitting code?** → Write access by an email to cryptographic_estimators@tii.ae.

- **Guidelines?** → Check CONTRIBUTING.md in the repository.

- **Don’t want to code?** →

Contributing

■ **Where?** → Public repository CryptographicEstimators³.

■ **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).

NIST PQC Signatures
Coverage

- Current included estimators:
Coverage

- Current included estimators:
  - Multivariate Quadratic (MQ)
  - Binary Syndrome Decoding (SD)
  - Permutation Equivalence (PE)
  - Linear Equivalence (LE)
  - Permuted Kernel (PK)

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

- Current included estimators:
  - Multivariate Quadratic (MQ) \( \rightarrow y = \mathcal{P}(x) \), with \( \mathcal{P} \) a quadratic map.
  - Binary Syndrome Decoding (SD) \( \rightarrow s = H \cdot e \), with \( \text{wt}(e) \leq \omega \).
  - Permutation Equivalence (PE) \( \rightarrow G' = SGQ \), with \( Q \) a monomial matrix.
  - Permuted Kernel (PK) \( \rightarrow 0 = H \cdot \pi \cdot x \), with \( \pi \) a permutation.

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

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

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

Current included estimators:

- Multivariate Quadratic (MQ) \(\rightarrow y = P(x)\), with \(P\) a quadratic map.
- Binary Syndrome Decoding (SD)
- Syndrome Decoding over \(\mathbb{F}_q\) (SDFq) \(\rightarrow s = H \cdot e\), with \(wt(e) \leq \omega\).
Coverage

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

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

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

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

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

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

Estimates for NIST Category I parameter sets

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

Thank you!
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.
- Regular Syndrome Decoding –> Marche Polytechnic University, TII, and potential other collaborators.

Thank you!
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.

Thank you!
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.