Complexity estimator for masking gadgets

Laurent Imbert[1], **Camille Mutschler**[1,2] & Thomas Roche[2]
[1]**LIRMM, Univ. Montpellier, CNRS, France**
[2]**NinjaLab, Montpellier, France**

October 19, 2023

Journées C2

Summary

Introduction to Kyber
- The NIST PQC Standardization Process and Kyber

Masking Kyber
- Introduction to Masking
- Masking Kyber
- The compression function

Complexity of the masked algorithms
- The problem of current gadget estimates
- Our Gadget Estimation Tool
- Some results

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

Kyber is:

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

Kyber is:

- A Key Encapsulation Mechanism

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

Kyber is:

- A Key Encapsulation Mechanism
- It's security is based on the lattice problem Learning With Errors (LWE)

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

Kyber is:

- A Key Encapsulation Mechanism
- It's security is based on the lattice problem Learning With Errors (LWE)
- It works with polynomials with 256 coefficients in $\mathbb{F}_q$, with $q = 3329$, a prime number

The NIST PQC Standardization Process

- **2016**: NIST Post-Quantum Standardization Process
- **July 2022**: Kyber chosen to be standardized

Kyber is:

- A Key Encapsulation Mechanism
- It's security is based on the lattice problem Learning With Errors (LWE)
- It works with polynomials with 256 coefficients in $\mathbb{F}_q$, with $q = 3329$, a prime number

Focus on ***Kyber's decapsulation***.
Information leaks $\Rightarrow$ must be **protected against side channel attacks**

Masking

Masking consist on **splitting a value into multiple shares** and transform the underlying circuit to process these shared variables securely.

Masking

Masking consist on **splitting a value into multiple shares** and transform the underlying circuit to process these shared variables securely.

Let $x \in \mathbb{F}_q$. Here is an example of first-order masking (n=2 shares):

Boolean masking:	$x = x_1 \oplus x_2$
Arithmetic masking:	$x \equiv x_1 + x_2 \pmod{q}$

We represent a masked value x as the n-tuple $(x_1, x_2, \ldots, x_n)$.

Masking gadgets

The masked variant of the circuits is called a **masking gadget**. Their complexity depends on the masking order.

Masking gadgets

The masked variant of the circuits is called a **masking gadget**. Their complexity depends on the masking order.

We have masking gadgets that allow us to compute the **addition** ($\mathcal{O}(n)$) and the **multiplication** ($\mathcal{O}(n^2)$) in a secure way.

Masking gadgets

The masked variant of the circuits is called a **masking gadget**. Their complexity depends on the masking order.

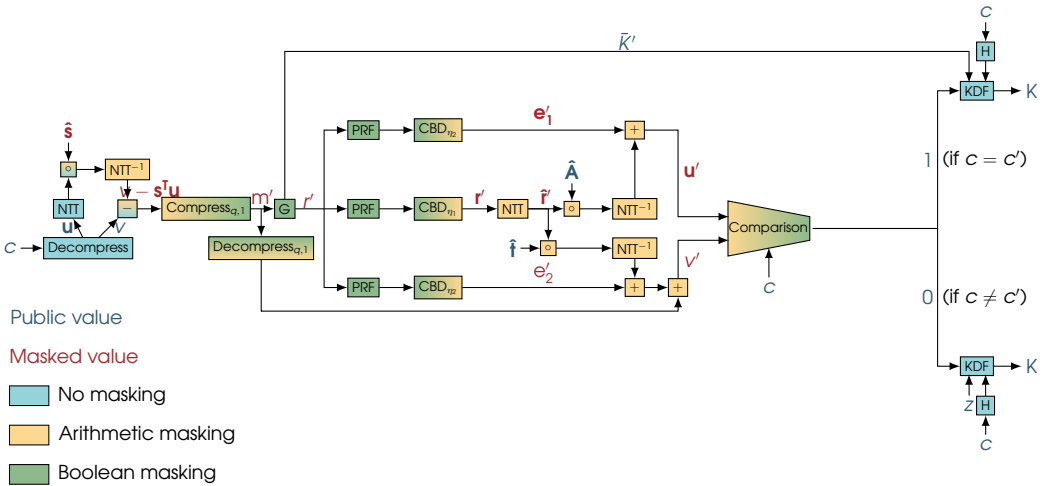
We have masking gadgets that allow us to compute the **addition** ($\mathcal{O}(n)$) and the **multiplication** ($\mathcal{O}(n^2)$) in a secure way.

There exists masking gadgets for **mask conversion**: $(x_1, x_2) \Rightarrow (x_1', x_2')$, such that:

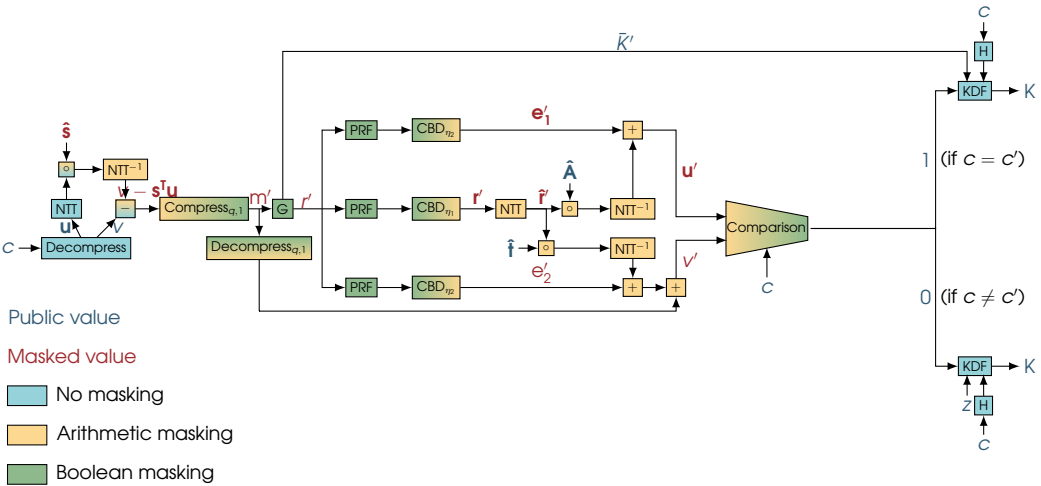
Boolean to arithmetic ($B2A$):	$x_1 \oplus x_2 = x$	$\Rightarrow$	$x_1' + x_2' \equiv x \pmod{q}$
Arithmetic to boolean ($A2B$):	$x_1 + x_2 \equiv x \pmod{q}$	$\Rightarrow$	$x_1' \oplus x_2' = x$

$\Rightarrow$ Conversions are very expensive, more than the multiplication.

Masking Kyber



Masking Kyber



One of the most expensive parts is the **compression** as it requires doing mask conversion.

The compression function

Let $x \in \mathbb{F}_q$:

$$\text{Compress}_{q,d}(x) = \left\lfloor \frac{2^d \cdot x}{q} \right\rceil \bmod 2^d$$

For $d = 1$ bit:

$$\text{Compress}_{q,1}(x) = \left\lfloor \frac{2 \cdot x}{q} \right\rceil \bmod 2 = \begin{cases} 1 & \text{if } \frac{q}{4} < x < \frac{3q}{4}, \\ 0 & \text{otherwise.} \end{cases}$$

The compression function

Let $x \in \mathbb{F}_q$:

$$\text{Compress}_{q,d}(x) = \left\lfloor \frac{2^d \cdot x}{q} \right\rceil \bmod 2^d$$

For $d = 1$ bit:

$$\text{Compress}_{q,1}(x) = \left\lfloor \frac{2 \cdot x}{q} \right\rceil \bmod 2 = \begin{cases} 1 & \text{if } \frac{q}{4} < x < \frac{3q}{4}, \\ 0 & \text{otherwise.} \end{cases}$$

- Hard to mask: we don't know how to perform a comparison in arithmetic masking
- Possible to do comparisons in boolean masking with some tricks

Compression masking gadgets

A number of masking gadgets have been proposed for masked compression. Each has its own characteristics:

- Some gadgets work only with $\mathbf{n = 2}$ **shares**, while others work with **any value of n**
- Some gadgets have been designed to do compression only to **1 bit**, while others have been designed to do compression to **d bits**, for all $d \in \mathbb{N}$
- Some gadgets use optimizations:
  - Table-based optimizations
  - Bitslicing

This wide variety of gadgets performing masked compression can make comparison difficult.

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

- Too high-level: asymptotic complexity, complexity in number of operations

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

- Too high-level: asymptotic complexity, complexity in number of operations
- Too low-level: implementations on specific chips

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

- Too high-level: asymptotic complexity, complexity in number of operations
- Too low-level: implementations on specific chips

Too high-level $\Rightarrow$ Do not give very interesting information in practice

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

- Too high-level: asymptotic complexity, complexity in number of operations
- Too low-level: implementations on specific chips

Too high-level $\Rightarrow$ Do not give very interesting information in practice

Exemple: Complexity of the masked compression in $\mathcal{O}(n^2 \log\log q)$
$\rightarrow n =$ number of shares (in practice 2 or 3 !)
$\rightarrow \log q = 12$ for Kyber

Complexity of the masked algorithms

There's a **large heterogeneity of complexity estimates** in the literature.

These estimates are either:

- Too high-level: asymptotic complexity, complexity in number of operations
- Too low-level: implementations on specific chips

Too high-level $\Rightarrow$ Do not give very interesting information in practice

Exemple: Complexity of the masked compression in $\mathcal{O}(n^2 \log\log q)$
$\rightarrow n =$ number of shares (in practice 2 or 3 !)
$\rightarrow \log q = 12$ for Kyber

Too low-level $\Rightarrow$ Not very comparable from one work to another

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

- Take into account the constraints of a component like:
  - Size of registers
  - Size of memory
  - Etc,...

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

- Take into account the constraints of a component like:
  - Size of registers
  - Size of memory
  - Etc,...
- Remaining sufficiently simple for a "high-level" comparison of the proposals

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

- Take into account the constraints of a component like:
  - Size of registers
  - Size of memory
  - Etc,...
- Remaining sufficiently simple for a "high-level" comparison of the proposals

For all the existing proposals in the literature, we want to:

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

- Take into account the constraints of a component like:
  - Size of registers
  - Size of memory
  - Etc,...
- Remaining sufficiently simple for a "high-level" comparison of the proposals

For all the existing proposals in the literature, we want to:

- Estimate the complexities of the algorithms with our new methods.
  Goal: develop a tool that estimates the complexities of all existing proposals with respect to a "simple" model of microcontroller.

Our idea for a better complexity estimation

Idea: Find the right level of abstraction to estimate the complexity of these algorithms, in order to:

- Take into account the constraints of a component like:
  - Size of registers
  - Size of memory
  - Etc,...
- Remaining sufficiently simple for a "high-level" comparison of the proposals

For all the existing proposals in the literature, we want to:

- Estimate the complexities of the algorithms with our new methods.
  Goal: develop a tool that estimates the complexities of all existing proposals with respect to a "simple" model of microcontroller.
- Provide a comparison of the existing masking methods on different "standard microcontroller"

Our Estimation Tool

To build our gadget estimation tool, we worked on:

Our Estimation Tool

To build our gadget estimation tool, we worked on:

- The representation of the gadgets

Our Estimation Tool

To build our gadget estimation tool, we worked on:

- The representation of the gadgets
- Defining the masking parameters
  - Number of shares
  - Size of the shares (in bits)

Our Estimation Tool

To build our gadget estimation tool, we worked on:

- The representation of the gadgets
- Defining the masking parameters
  - Number of shares
  - Size of the shares (in bits)
- Modeling a "microcontroller"
  - Memory space (in bits)
  - Sum of register space (in bits)
  - Cost of atomic operations in CPU cycles (Assembly instructions, random value generation, etc.)

Our Estimation Tool

To build our gadget estimation tool, we worked on:

- The representation of the gadgets
- Defining the masking parameters
  - Number of shares
  - Size of the shares (in bits)
- Modeling a "microcontroller"
  - Memory space (in bits)
  - Sum of register space (in bits)
  - Cost of atomic operations in CPU cycles (Assembly instructions, random value generation, etc.)

We have associated 2 types of estimation functions with each gadget:

- a memory cost estimation function
- a performance estimation function

Memory cost estimation function

Aim:

- Take into account the critical memory path of the memory to estimate the maximum memory space the gadget will need to run

This function is used to:

- Check that the microcontroller we've modeled has enough memory to run the gadget
- Check whether the data manipulated in the gadget can be stored in registers or whether they must be stored in memory

If data has to be stored in memory, this can mean additional performance costs due the use of load and store operations.

Performance estimation functions

Aim:

- Estimate the cost of a gadget based on the CPU cycles assigned to each atomic operation performed by the gadget and linked to the microcontroller on which it is to be estimated

Performance estimation functions

Aim:

- Estimate the cost of a gadget based on the CPU cycles assigned to each atomic operation performed by the gadget and linked to the microcontroller on which it is to be estimated

The particularity of this estimation:

- Some negligible costs are not included in our estimations
  - Ex: Loop operations such as incrementing a counter
- We've built all our performance estimation functions in the same way to ensure consistency and comparability between our gadgets

Performance estimation functions

Aim:

- Estimate the cost of a gadget based on the CPU cycles assigned to each atomic operation performed by the gadget and linked to the microcontroller on which it is to be estimated

The particularity of this estimation:

- Some negligible costs are not included in our estimations
  - Ex: Loop operations such as incrementing a counter
- We've built all our performance estimation functions in the same way to ensure consistency and comparability between our gadgets

Since we don't take into account all operations, we say that our estimations are calculated in CPU cycle equivalent (CCE).

Example: Table-based A2B conversion

Gadget: Secure A2B-1bit conversion from (CGMZ22), on 6-bit inputs.
Sum of register space: 416 bits.

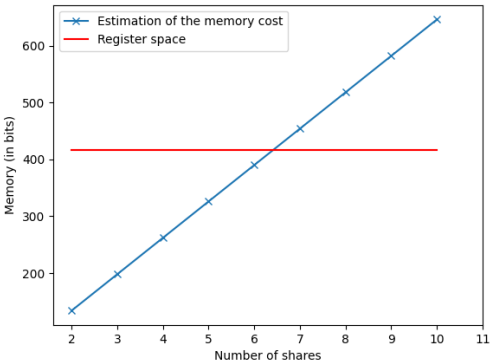


Figure: Memory cost estimation

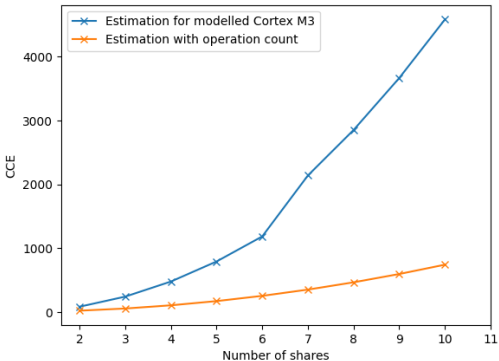


Figure: Performance estimation

Comparison of several compression gadgets

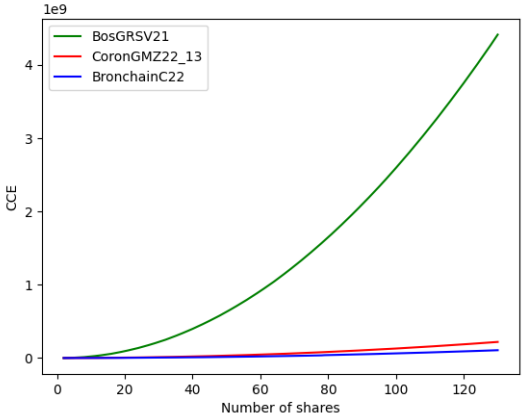


Figure: Performance estimation of various Masked Compression (with rand generation = 32, for 256 coefficients), on a modelled Cortex M3

Complexity:

- BosGRSV21: $\mathcal{O}(n^2 \log_2(\log_2 q))$
- CoronGMZ22_13: $\mathcal{O}(n^2)$
- BronchainC22: $\mathcal{O}(\lceil \log_2(q \cdot n) \rceil n^2)$

Comparison of several compression gadgets

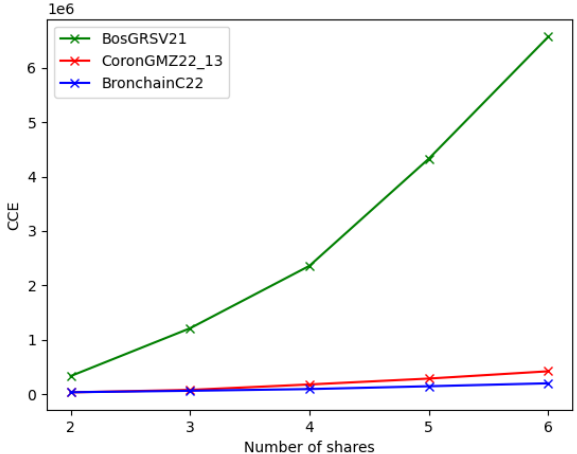


Figure: A closer look at the two most effective masked compression gadgets in the literature

Conclusion

This tool will be publicly available.

- It will be possible for everyone to work on it, and give us feedback on things that could be improved

Conclusion

This tool will be publicly available.

- It will be possible for everyone to work on it, and give us feedback on things that could be improved

Limitations and idea for future improvements:

- It's difficult to make our estimates homogeneous for all gadgets with the way we've chosen to model a microcontroller and build our estimation functions
- Take into account gadget security (NI, SNI, PINI) to estimate the security of a gadget and evaluate gadget composability

Conclusion

This tool will be publicly available.

- It will be possible for everyone to work on it, and give us feedback on things that could be improved

Limitations and idea for future improvements:

- It's difficult to make our estimates homogeneous for all gadgets with the way we've chosen to model a microcontroller and build our estimation functions
- Take into account gadget security (NI, SNI, PINI) to estimate the security of a gadget and evaluate gadget composability

Conclusion

This tool will be publicly available.

- It will be possible for everyone to work on it, and give us feedback on things that could be improved

Limitations and idea for future improvements:

- It's difficult to make our estimates homogeneous for all gadgets with the way we've chosen to model a microcontroller and build our estimation functions
- Take into account gadget security (NI, SNI, PINI) to estimate the security of a gadget and evaluate gadget composability

Last but not least:

- We're currently looking for a name for our estimating tool, so if you have any ideas, please let us know! :)

Thank you for your attention !

Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order table-based conversion algorithms and masking lattice-based encryption.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2022(2):1–40, 2022.

Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto.
In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 534–564. Springer, 2019.