Adaptive versus Static Multi-oracle Algorithms, and Quantum Security of a Split-key PRF

Jelle Don[1] Serge Fehr[1,2] **Yu-Hsuan Huang**[1]

[1]Centrum Wiskunde & Informatica, The Netherlands

[2]Leiden University, The Netherlands

Overview

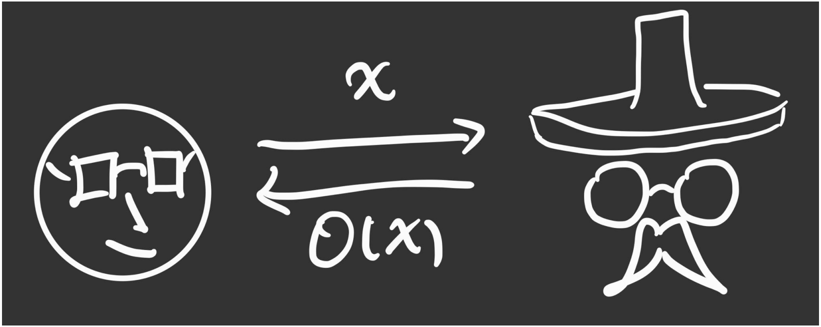
- Adaptive versus Static Multi-oracle Algorithms
- Our Results
  - Adaptive-to-static Compiler
  - Quantum Security of a skPRF
- Summary

Adaptive versus Static Multi-oracle Algorithms

Oracle Algorithms

An algorithm $\mathcal{A}^{\mathcal{O}}$ querying a (possibly randomized) function $\mathcal{O}$ for free.

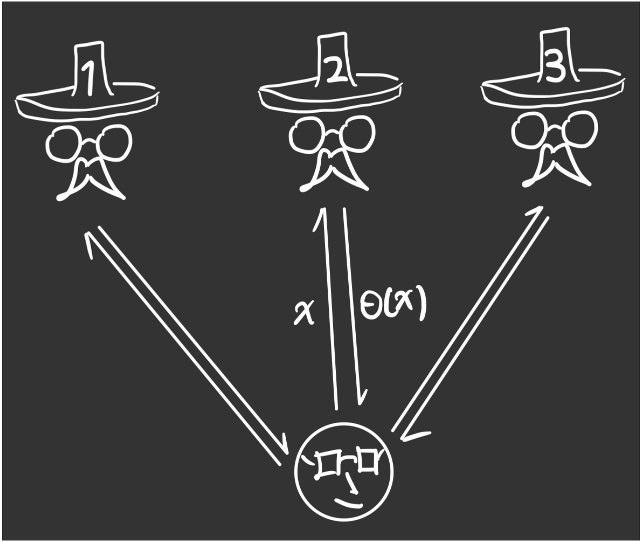
Assumption: a fixed upper bound q on #queries to $\mathcal{O}$.



Multi-oracle Algorithms

An algorithm $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$ querying multiple functions $\mathcal{O}_1,\ldots,\mathcal{O}_n$ for free.

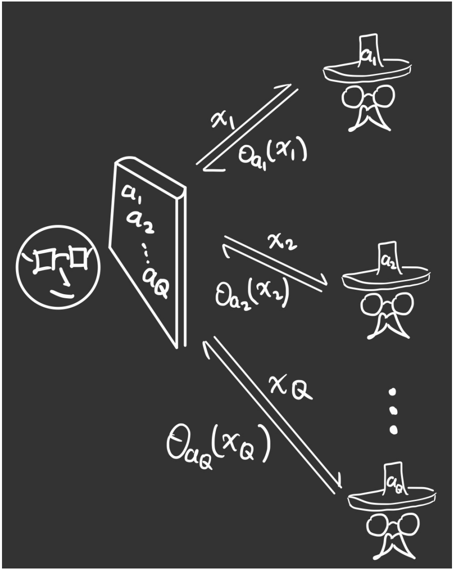
Assumption: fixed upper bounds $q_1,\ldots,q_n$ on #queries to $\mathcal{O}_1,\ldots,\mathcal{O}_n$.



Static Multi-oracle Algorithms

A multi-oracle algorithm with predetermined querying order.

In contrast, an adaptive algorithm can decide which oracle to query at what point dependent on previous oracle responses.

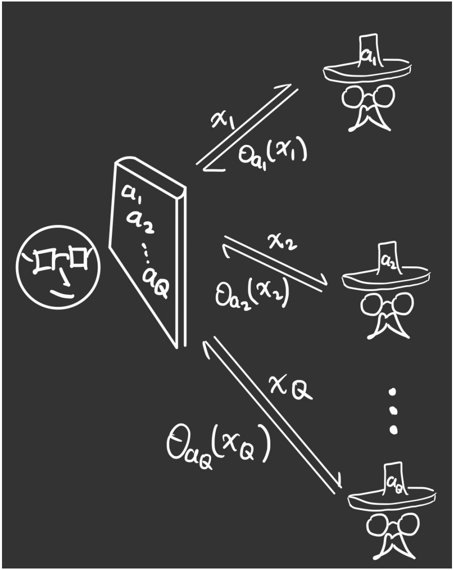


Static Multi-oracle Algorithms

A multi-oracle algorithm with predetermined querying order.

In contrast, an adaptive algorithm can decide which oracle to query at what point dependent on previous oracle responses.

Why static algorithms?



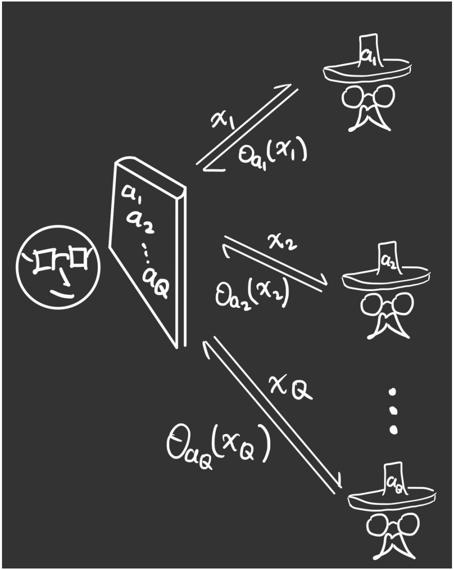
Static Multi-oracle Algorithms

A multi-oracle algorithm with predetermined querying order.

In contrast, an adaptive algorithm can decide which oracle to query at what point dependent on previous oracle responses.

Why static algorithms?

- easier for analysis



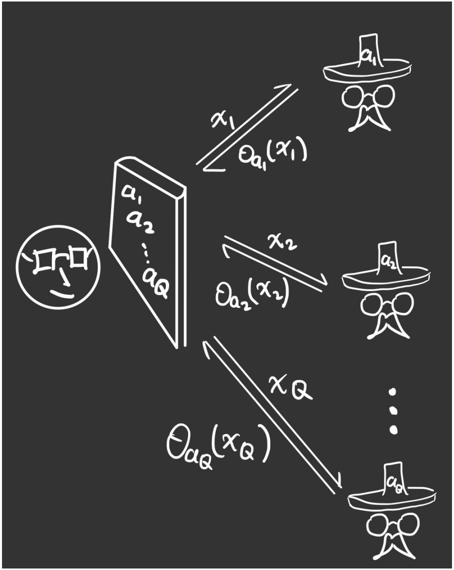
Static Multi-oracle Algorithms

A multi-oracle algorithm with predetermined querying order.

In contrast, an adaptive algorithm can decide which oracle to query at what point dependent on previous oracle responses.

Why static algorithms?

- easier for analysis
- (sometimes) better bounds



Example: Multi-oracle Algorithms

Attackers $\mathcal{A}$ against cryptographic schemes in the random oracle model:

- Encryption/KEM: $\mathcal{O}_1 =$ random oracle and $\mathcal{O}_2 =$ decrpt/decap oracle.
- Signature: $\mathcal{O}_1 =$ random oracle and $\mathcal{O}_2 =$ signing oracle.
- Pseudorandom function: $\mathcal{O}_1 =$ random oracle and $\mathcal{O}_2 =$ evaluation oracle.

Our Results

Our result consists of two parts

In the first part, we give a black-box, straight-line, efficient compiler transforming **any** (classical or quantum) multi-oracle algorithm $\mathcal{A}$ to a **static** one $\mathcal{B}[\mathcal{A}]$, with a **mild blow-up on its query complexity**.

Our Results

Our result consists of two parts

In the first part, we give a black-box, straight-line, efficient compiler transforming **any** (classical or quantum) multi-oracle algorithm $\mathcal{A}$ to a **static** one $\mathcal{B}[\mathcal{A}]$, with a **mild blow-up on its query complexity**.

- $\mathcal{A}$ makes $q_1, \ldots, q_n$ respective queries to $\mathcal{O}_1, \ldots, \mathcal{O}_n$

Our Results

Our result consists of two parts

In the first part, we give a black-box, straight-line, efficient compiler transforming **any** (classical or quantum) multi-oracle algorithm $\mathcal{A}$ to a **static** one $\mathcal{B}[\mathcal{A}]$, with a **mild blow-up on its query complexity**.

- $\mathcal{A}$ makes $q_1, \ldots, q_n$ respective queries to $\mathcal{O}_1, \ldots, \mathcal{O}_n$
  $\Rightarrow \mathcal{B}[\mathcal{A}](1^{q_1}, \ldots, 1^{q_n})$ makes $nq_1, \ldots, nq_n$ respective queries only.

Our Results

Our result consists of two parts

In the first part, we give a black-box, straight-line, efficient compiler transforming **any** (classical or quantum) multi-oracle algorithm $\mathcal{A}$ to a **static** one $\mathcal{B}[\mathcal{A}]$, with a **mild blow-up on its query complexity**.

- $\mathcal{A}$ makes $q_1, \ldots, q_n$ respective queries to $\mathcal{O}_1, \ldots, \mathcal{O}_n$

  $\Rightarrow \mathcal{B}[\mathcal{A}](1^{q_1}, \ldots, 1^{q_n})$ makes $nq_1, \ldots, nq_n$ respective queries only.
- Applications: simplifying existing results [ABB$^+$17, ABKM21] but also obtaining an enhanced bound [JST21].

Our Results

Our result consists of two parts

In the first part, we give a black-box, straight-line, efficient compiler transforming **any** (classical or quantum) multi-oracle algorithm $\mathcal{A}$ to a **static** one $\mathcal{B}[\mathcal{A}]$, with a **mild blow-up on its query complexity**.

- $\mathcal{A}$ makes $q_1, \ldots, q_n$ respective queries to $\mathcal{O}_1, \ldots, \mathcal{O}_n$
  $\Rightarrow \mathcal{B}[\mathcal{A}](1^{q_1}, \ldots, 1^{q_n})$ makes $nq_1, \ldots, nq_n$ respective queries only.
- Applications: simplifying existing results [ABB$^+$17, ABKM21] but also obtaining an enhanced bound [JST21].

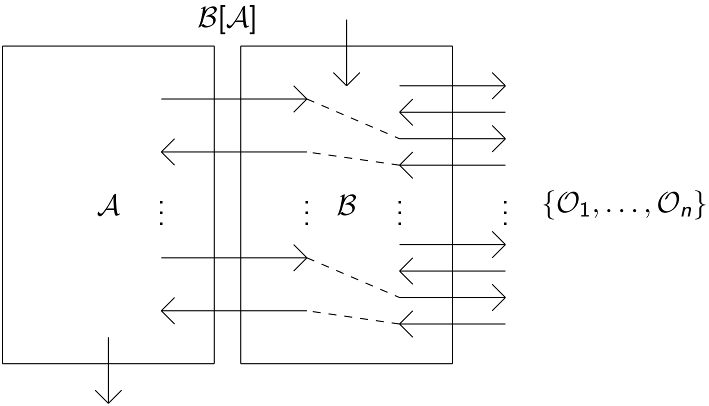
In the second part, we show the **QROM security** of a particularly efficient skPRF by Giacon, Heuer and Poettering [GHP18].

- Consequently, an efficient **KEM combiner** is QROM-secure.
- Our analysis crucially relies on the abovementioned compiler.

Part 1: Adaptive-to-static Compiler

Our Results: The Adaptive-to-static Compiler

Our compiler works by running an *interactive oracle algorithm* $\mathcal{B}$ as an interface between $\mathcal{A}$ and oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$ and re-routing the adaptive queries to the pre-determined static ones.



A Naive Compiler

Consider $n = 2$.
Suppose $\mathcal{A}$ makes q_1, q_2 queries to $\mathcal{O}_1, \mathcal{O}_2$ respectively.

- Let $\mathcal{B}^{\text{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ query in order

$$(\mathcal{O}_1\mathcal{O}_2)^{q_1+q_2} := \underbrace{(\mathcal{O}_1\mathcal{O}_2)\ldots(\mathcal{O}_1\mathcal{O}_2)}_{q_1+q_2 \text{ times}} .$$

- Forward the query of $\mathcal{A}$ and do a dummy query for mis-match.

A Naive Compiler

Consider $n = 2$.
Suppose $\mathcal{A}$ makes q_1, q_2 queries to $\mathcal{O}_1, \mathcal{O}_2$ respectively.

- Let $\mathcal{B}^{\text{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ query in order

$$(\mathcal{O}_1\mathcal{O}_2)^{q_1+q_2} := \underbrace{(\mathcal{O}_1\mathcal{O}_2)\dots(\mathcal{O}_1\mathcal{O}_2)}_{q_1+q_2 \text{ times}} .$$

- Forward the query of $\mathcal{A}$ and do a dummy query for mis-match.

$\mathcal{B}^{\text{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ makes $q_1 + q_2$ queries to both $\mathcal{O}_1, \mathcal{O}_2$:

A Naive Compiler

Consider $n = 2$.
Suppose $\mathcal{A}$ makes q_1, q_2 queries to $\mathcal{O}_1, \mathcal{O}_2$ respectively.

- Let $\mathcal{B}^{\text{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ query in order

$$(\mathcal{O}_1\mathcal{O}_2)^{q_1+q_2} := \underbrace{(\mathcal{O}_1\mathcal{O}_2)\dots(\mathcal{O}_1\mathcal{O}_2)}_{q_1+q_2 \text{ times}} .$$

- Forward the query of $\mathcal{A}$ and do a dummy query for mis-match.

$\mathcal{B}^{\text{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ makes $q_1 + q_2$ queries to both $\mathcal{O}_1, \mathcal{O}_2$:

- What if $q_1 = q_2^2$? Then it makes $\approx q_1 >> q_2$ queries to **both**.

A Naive Compiler

Consider $n = 2$.

Suppose $\mathcal{A}$ makes q_1, q_2 queries to $\mathcal{O}_1, \mathcal{O}_2$ respectively.

- Let $\mathcal{B}^{\mathsf{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ query in order

$$(\mathcal{O}_1\mathcal{O}_2)^{q_1+q_2} := \underbrace{(\mathcal{O}_1\mathcal{O}_2)\ldots(\mathcal{O}_1\mathcal{O}_2)}_{q_1+q_2 \text{ times}} .$$

- Forward the query of $\mathcal{A}$ and do a dummy query for mis-match.

$\mathcal{B}^{\mathsf{naive}}[\mathcal{A}](1^{q_1}, 1^{q_2})$ makes $q_1 + q_2$ queries to both $\mathcal{O}_1, \mathcal{O}_2$:

- What if $q_1 = q_2^2$? Then it makes $\approx q_1 >> q_2$ queries to **both**.
  We want $\approx q_1$ queries to $\mathcal{O}_1$ and $\approx q_2$ queries to $\mathcal{O}_2$ instead!!!

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\}\ .$$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in \mathit{Char}(q_1, q_2)$ where

$$\mathit{Char}(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in \mathit{Char}(q_1 + q_2, q_1 + q_2)$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \enspace .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$

Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$
  - $s' = 2221 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in \mathit{Char}(q_1, q_2)$ where

$$\mathit{Char}(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in \mathit{Char}(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $\mathit{Char}(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in \mathit{Char}(1, 6) \subseteq \mathit{Char}(2q_1, 2q_2)$
  - $s' = 2212 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in \mathit{Char}(q_1, q_2)$ where

$$\mathit{Char}(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in \mathit{Char}(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $\mathit{Char}(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in \mathit{Char}(1, 6) \subseteq \mathit{Char}(2q_1, 2q_2)$
  - $s' = 2122 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\dots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\}$.

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$
  - $s' = 1222 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\dots12$ is a supersequence of every $s' \in \mathit{Char}(q_1, q_2)$ where

$\mathit{Char}(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\}$.

Problem of naive construction: $s \in \mathit{Char}(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $\mathit{Char}(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in \mathit{Char}(1, 6) \subseteq \mathit{Char}(2q_1, 2q_2)$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$
- for $(q_1, q_2) = (3, 10)$, pick $s = 2221222122221222122212222$
  - $s \in Char(6, 19) \subseteq Char(2q_1, 2q_2)$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in \mathit{Char}(q_1, q_2)$ where

$$\mathit{Char}(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} .$$

Problem of naive construction: $s \in \mathit{Char}(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $\mathit{Char}(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in \mathit{Char}(1, 6) \subseteq \mathit{Char}(2q_1, 2q_2)$
- for $(q_1, q_2) = (3, 10)$, pick $s = 2221222122212221222122222$
  - $s \in \mathit{Char}(6, 19) \subseteq \mathit{Char}(2q_1, 2q_2)$
  - $1222212222122 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\dots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$
- for $(q_1, q_2) = (3, 10)$, pick $s = 2221222122221222122212222$
  - $s \in Char(6, 19) \subseteq Char(2q_1, 2q_2)$
  - $1112222222222 \sqsubseteq s$

A Combinatorial Approach

Abstract formulation: the string $s = (12)^{q_1+q_2} = 1212\ldots12$ is a supersequence of every $s' \in Char(q_1, q_2)$ where

$$Char(q_1, q_2) := \{s' \in \{1,2\}^* : \text{every } \sigma \in \{1,2\} \text{ occurs in } s' \text{ for } q_\sigma \text{ times}\} \ .$$

Problem of naive construction: $s \in Char(q_1 + q_2, q_1 + q_2)$
Goal: find such s in, say $Char(2q_1, 2q_2)$.

Example

- for $(q_1, q_2) = (1, 3)$, pick $s = 2221222$
  - $s \in Char(1, 6) \subseteq Char(2q_1, 2q_2)$
- for $(q_1, q_2) = (3, 10)$, pick $s = 2221222122221222122212222$
  - $s \in Char(6, 19) \subseteq Char(2q_1, 2q_2)$

Our Embedding Lemma

Let $(q_1, \ldots, q_n) \in \mathbb{N}^n$.

Lemma

There exists a string $s \in Char(nq_1, \ldots, nq_n)$ such that every string $s' \in Char(q_1, \ldots, q_n)$ is a subsequence of s.

Our Embedding Lemma

Let $(q_1, \ldots, q_n) \in \mathbb{N}^n$.

Lemma

There exists a string $s \in Char(nq_1, \ldots, nq_n)$ such that every string $s' \in Char(q_1, \ldots, q_n)$ is a subsequence of s.

Furthermore, such s is polynomial-time computable given $(1^{q_1}, \ldots, 1^{q_n})$ in unary representation.

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \xleftarrow{\text{init}} \epsilon$

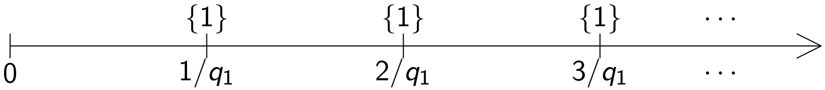


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \xleftarrow{\text{init}} \epsilon$

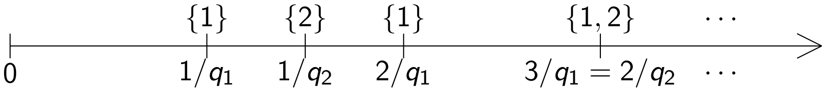


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| 1 = 1$

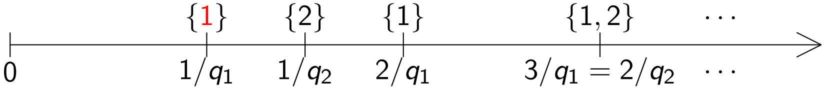


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| 2 = 12$

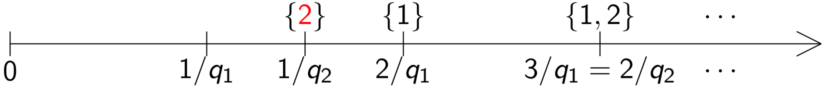


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| 1 = 121$

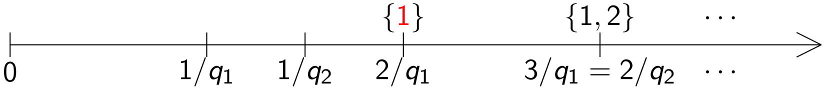


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| 1 = 1211$

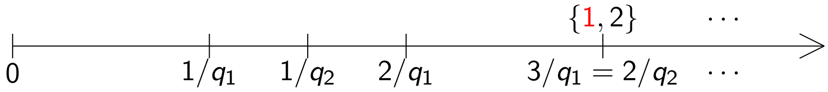


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| 2 = 12112$

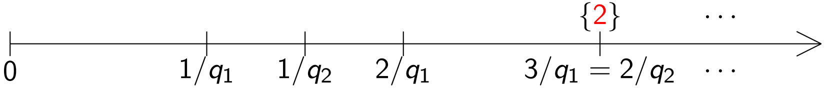


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| \ldots = 12112 \ldots$

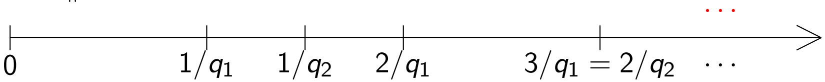


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s \leftarrow s \| \ldots = 12112\ldots$ until we reach time n

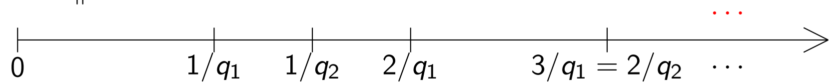


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Our Embedding Lemma

Proof.

Idea: distribute each symbol $\sigma \in [n]$ evenly within the interval $(0, n]$ and collect them from left to right.

$s = 12112\ldots$ until reach time n

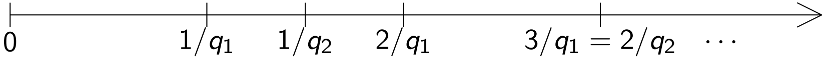


Figure: Constructing the string s (here with $3/q_1 = 2/q_2$)

$\square$

Part 2: Quantum-security of a skPRF

Main Applications: skPRF

A skPRF is a function $\mathcal{F}(k_1, \ldots, k_n, x)$ such that:

- for each i: $\mathcal{F}$ is pseudorandom as a function with key k_i.
  (technical constraint: attacker never query the same x twice)
- Implication: skPRF $\Rightarrow$ KEM-combiner [GHP18]

Main Applications: skPRF

A skPRF is a function $\mathcal{F}(k_1, \ldots, k_n, x)$ such that:

- for each i: $\mathcal{F}$ is pseudorandom as a function with key k_i.
  (technical constraint: attacker never query the same x twice)
- Implication: skPRF $\Rightarrow$ KEM-combiner [GHP18]

Efficient hash-based instantiation by [GHP18]:

$$\mathcal{F}(k_1, \ldots, k_n, x) := H(g(k_1, \ldots, k_n), x) \text{ for “key-mixing” } g \,.$$

Already proven **classically** secure, **quantum** security unknown.

Main Applications: skPRF

A skPRF is a function $\mathcal{F}(k_1, \ldots, k_n, x)$ such that:

- for each i: $\mathcal{F}$ is pseudorandom as a function with key k_i.
  (technical constraint: attacker never query the same x twice)
- Implication: skPRF $\Rightarrow$ KEM-combiner [GHP18]

Efficient hash-based instantiation by [GHP18]:

$$\mathcal{F}(k_1, \ldots, k_n, x) := H(g(k_1, \ldots, k_n), x) \text{ for “key-mixing” } g \; .$$

Already proven **classically** secure, **quantum** security unknown.

Theorem (Our result: quantum security of $\mathcal{F}$)

In the QROM, any skPRF attacker with at most q_F, q_H respective queries to $\mathcal{F}, H$ has advantage at most $4q_H\sqrt{2q_F\epsilon} + 4q_F\sqrt{2q_H\epsilon}$.

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \ldots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \ldots H'}^{q_{H,2}} \mathcal{F} H' \ldots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F \epsilon}$

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F \epsilon}$
- summing up, loss: $\sum_i 2q_{H,i}\sqrt{q_F \epsilon}$

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F \epsilon}$
- summing up, loss: $\sum_i 2q_{H,i}\sqrt{q_F \epsilon}$
- without any compiling, $q_{H,i} \leq q_H$ gives

$$q_H(q_F + 1)\sqrt{q_F \epsilon}$$

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F\epsilon}$
- summing up, loss: $\sum_i 2q_{H,i}\sqrt{q_F\epsilon}$
- without any compiling, $q_{H,i} \leq q_H$ gives

$$q_H(q_F + 1)\sqrt{q_F\epsilon}$$

- naive compiler: $\sum_i q_{H,i} \leq q_H + q_F$ and q_F becoming $q_F + q_H$ gives

$$2(q_H + q_F)\sqrt{(q_H + q_F)\epsilon}$$

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F \epsilon}$
- summing up, loss: $\sum_i 2q_{H,i}\sqrt{q_F \epsilon}$
- without any compiling, $q_{H,i} \leq q_H$ gives

$$q_H(q_F + 1)\sqrt{q_F \epsilon}$$

- naive compiler: $\sum_i q_{H,i} \leq q_H + q_F$ and q_F becoming $q_F + q_H$ gives

$$2(q_H + q_F)\sqrt{(q_H + q_F)\epsilon}$$

- our compiler: $\sum_i q_{H,i} \leq 2q_H$ and factor 2 blow-up on q_F, gives

$$4q_H\sqrt{2q_F \epsilon}$$

Quantum-security of a skPRF

Proof idea (for random function R and auxiliary oracle H'):

- initial querying pattern: $\overbrace{H' \dots H'}^{q_{H,1}} \mathcal{F} \overbrace{H' \dots H'}^{q_{H,2}} \mathcal{F} H' \dots$
- from left to right, replace every H' to H and every $\mathcal{F}$ to R

Let's look at the losses for replacing H' to H:

- The loss replacing H' to H in each block: $2q_{H,i}\sqrt{q_F \epsilon}$
- summing up, loss: $\sum_i 2q_{H,i}\sqrt{q_F \epsilon}$
- without any compiling, $q_{H,i} \leq q_H$ gives
$$q_H(q_F + 1)\sqrt{q_F \epsilon}$$
- naive compiler: $\sum_i q_{H,i} \leq q_H + q_F$ and q_F becoming $q_F + q_H$ gives
$$2(q_H + q_F)\sqrt{(q_H + q_F)\epsilon}$$
- our compiler: $\sum_i q_{H,i} \leq 2q_H$ and factor 2 blow-up on q_F, gives
$$4q_H\sqrt{2q_F \epsilon}$$

Our proof crucially relies on the compiler. ✓

Summary

Summary

Our result consists of two parts

In the first part, we give a compiler transforming a multi-oracle algorithm $\mathcal{A}$ with $(q_1, \ldots, q_n)$ queries to a static one with $(nq_1, \ldots, nq_n)$ queries.

- simplifying existing results [ABB$^+$17, ABKM21] but also obtaining an enhanced bound [JST21].

Summary

Our result consists of two parts

In the first part, we give a compiler transforming a multi-oracle algorithm $\mathcal{A}$ with $(q_1, \ldots, q_n)$ queries to a static one with $(nq_1, \ldots, nq_n)$ queries.

- simplifying existing results [ABB+17, ABKM21] but also obtaining an enhanced bound [JST21].

In the second part, we give the QROM security of the hash-based skPRF constructed by Giacon, Heuer and Poettering [GHP18].

- Consequently, the KEM combiner using $\mathcal{F}$ is QROM-secure.
- Our analysis crucially relies on the abovementioned compiler.

Summary

Our result consists of two parts

In the first part, we give a compiler transforming a multi-oracle algorithm $\mathcal{A}$ with $(q_1, \ldots, q_n)$ queries to a static one with $(nq_1, \ldots, nq_n)$ queries.

- simplifying existing results [ABB+17, ABKM21] but also obtaining an enhanced bound [JST21].

In the second part, we give the QROM security of the hash-based skPRF constructed by Giacon, Heuer and Poettering [GHP18].

- Consequently, the KEM combiner using $\mathcal{F}$ is QROM-secure.
- Our analysis crucially relies on the abovementioned compiler.

Take away: if you have adaptive adversaries, use our compiler!

That's It

Thanks for your listening!

Arxiv. *2206.08132*
Eprint. *2022/773*

References I

[ABB+17] Erdem Alkim, Nina Bindel, Johannes Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting TESLA in the quantum random oracle model. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 143–162. Springer, 2017.

[ABKM21] Gorjan Alagic, Chen Bai, Jonathan Katz, and Christian Majenz. Post-quantum security of the Even-Mansour cipher. Cryptology ePrint Archive, Report 2021/1601, 2021. `https://ia.cr/2021/1601`.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In *IACR International Workshop on Public Key Cryptography*, pages 190–218. Springer, 2018.

References II

[JST21] Joseph Jaeger, Fang Song, and Stefano Tessaro. Quantum key-length extension. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography Conference*, pages 209–239. Springer, 2021.