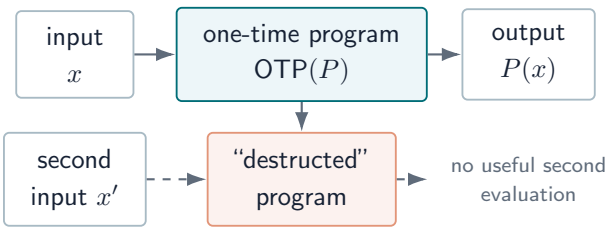
On Best-Possible One-Time Programs

Aparna Gupte, Jiahui Liu, *Luowen Qian*, Justin Raizes, Bhaskar Roberts, Mark Zhandry

One-Time Programs (OTPs) [Goldwasser–Kalai–Rothblum'08]

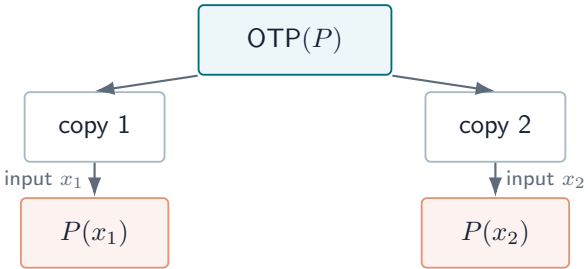
Ideal goal

A one-time program for P lets a user evaluate P on one input of their choice, while preventing them from learning anything else useful about P.



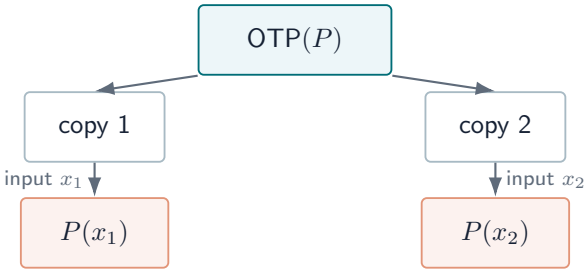
Classical OTPs are impossible

Suppose OTP outputs classical bitstrings.



Classical OTPs are impossible

Suppose OTP outputs classical bitstrings.

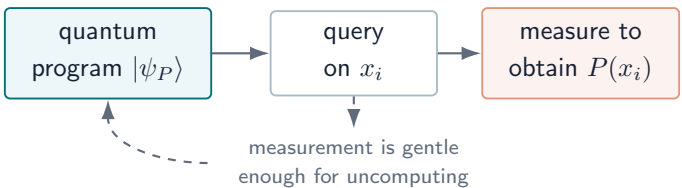


No cloning please rescue 🙏🙏🙏

Quantum OTPs are also impossible

Broadbent–Gutoski–Stebila'13 impossibility

Correctness for deterministic classical $P \Rightarrow$ measurement of output is gentle

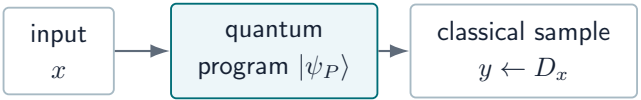


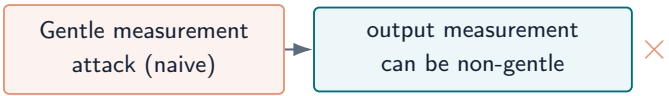
Even worse: coherent access for free

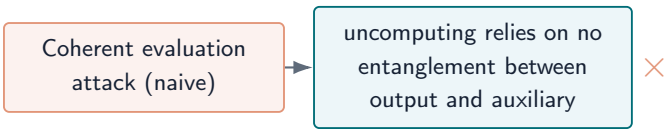
Coherent computation is always possible even if implemented by a quantum state:

$$|x\rangle|\psi_P\rangle \longmapsto |x\rangle|P(x)\rangle|\psi_P\rangle.$$

A way out? Quantum program + randomized outputs



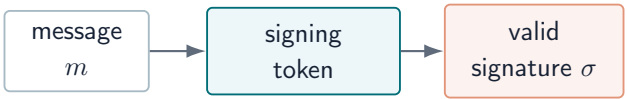




Signature tokens [Ben-David–Sattath'19]

A signature token is a (quantum) one-time program for the signing functionality:

$$\mathsf{Sign}_{\mathsf{sk}}(m; r) \to \sigma \quad \text{with} \quad \mathsf{Ver}_{\mathsf{vk}}(m, \sigma) = 1.$$



Security: an adversary should not produce valid signatures on two different messages.

Prior work: randomized OTPs

Functionalities	Prior work	Security achieved	Gap
Signing	Ben-David–Sattath'19	Signature tokens	Only signing

For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

Prior work: randomized OTPs

Functionalities	**Prior work**	**Security achieved**	**Gap**
Signing	Ben-David–Sattath'19	Signature tokens	Only signing
High-entropy randomized $P(x;r)$	Gunn–Movassagh'25	Gentle-measurement-type attack mitigated	Unsatisfactory security

For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

Prior work: randomized OTPs

Functionalities	**Prior work**	**Security achieved**	**Gap**
Signing	Ben-David–Sattath'19	Signature tokens	Only signing
High-entropy randomized $P(x; r)$	Gunn–Movassagh'25	Gentle-measurement-type attack mitigated	Unsatisfactory security
General randomized $P(x; r)$	Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25	CSEQ-secure compilers	Meaning of CSEQ?

For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

Prior work: randomized OTPs

Functionalities	**Prior work**	**Security achieved**	**Gap**
Signing	Ben-David–Sattath'19	Signature tokens	Only signing
High-entropy randomized $P(x;r)$	Gunn–Movassagh'25	Gentle-measurement-type attack mitigated	Unsatisfactory security
General randomized $P(x;r)$	Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25	CSEQ-secure compilers	Meaning of CSEQ?
PRF/NIZK		Two-time indistinguishability/unprovability	Application-specialized security

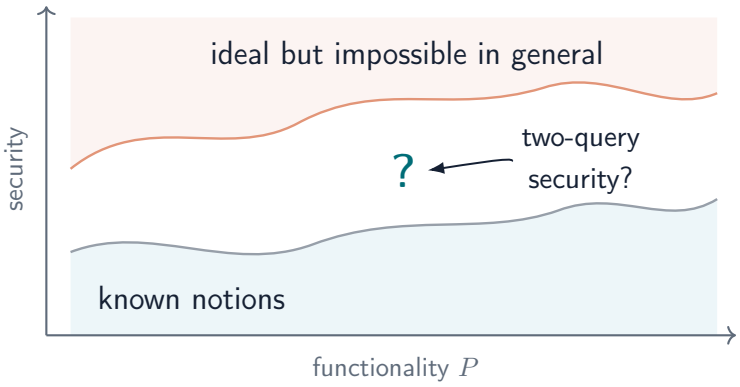
For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

Prior work: randomized OTPs

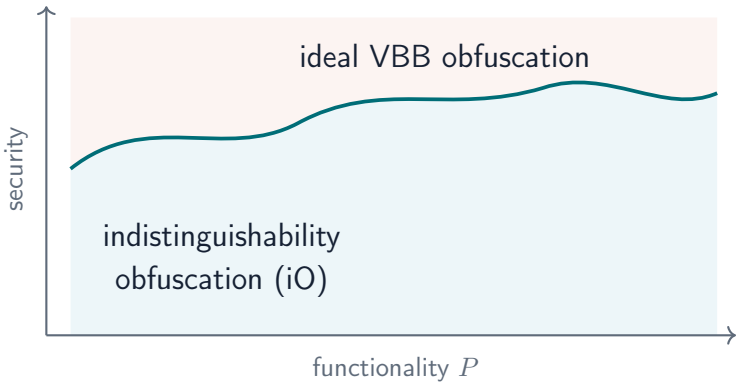
Functionalities	**Prior work**	**Security achieved**	**Gap**
Signing	Ben-David–Sattath'19	Signature tokens	Only signing
High-entropy randomized $P(x;r)$	Gunn–Movassagh'25	Gentle-measurement-type attack mitigated	Unsatisfactory security
General randomized $P(x;r)$	Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25	CSEQ-secure compilers	Meaning of CSEQ?
PRF/NIZK		Two-time indistinguishability/unprovability	Application-specialized security
Contrived high-entropy randomized $P(x;r)$		Ideal one-time interface $x \mapsto y \leftarrow D_x$ is impossible	(negative result)

For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

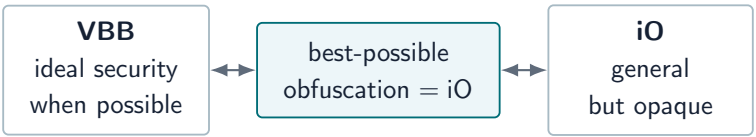
What is the strongest achievable security?



Many-time analogy: obfuscation



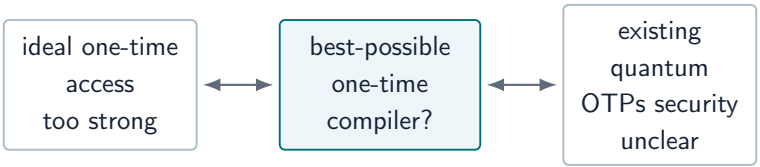
Best-Possible Obfuscation [Goldwasser–Rothblum'07]



If a functionality can be VBB-obfuscated, then iO achieves VBB.
(Also applies to other obfuscation security notions.)
It suffices to focus on constructing iO!

Best-Possible One-Time Programs?

Can quantum OTPs have the same "best-possible" story?



Is there a generic one-time compiler that achieves the best-possible security for any functionality?

Talk outline

- Background and motivation
  (you are here)
- Best-possible one-time programs are impossible
  - Proof sketch
- Getting around the impossibility
  - More new one-time security notions
  - Generalizing to quantum functionalities
  - New OTP constructions

Theorem 1: no best-possible OTP

Assuming lossy encryptions exist, no best-possible one-time compiler exists.

- Lossy encryptions can be constructed from Learning with Errors (LWE) or from weak pseudorandom group actions.
- Also relativizes to all oracles.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

Claim: There is a one-time compiler for D that achieves ideal one-time security.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

Claim: There is a one-time compiler for D that achieves ideal one-time security.

Construction: $\mathsf{OTP}^*(D)$ samples $y \leftarrow D(0)$ and outputs a constant-program that always outputs y.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

Claim: There is a one-time compiler for D that achieves ideal one-time security.

Construction: $\mathsf{OTP}^*(D)$ samples $y \leftarrow D(0)$ and outputs a constant-program that always outputs y.

- This achieves ideal one-time security: $\mathsf{OTP}^*(D)$ is simulatable by one query.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

Claim: There is a one-time compiler for D that achieves ideal one-time security.

Construction: $\mathsf{OTP}^*(D)$ samples $y \leftarrow D(0)$ and outputs a constant-program that always outputs y.

- This achieves ideal one-time security: $\mathsf{OTP}^*(D)$ is simulatable by one query.
- But this construction is one-time correct only for constant-distribution functionalities.

First step: a trivial one-time program

Consider $D(x)$ samples from a fixed distribution C, independent of x.

Claim: There is a one-time compiler for D that achieves ideal one-time security.

Construction: $\mathsf{OTP}^*(D)$ samples $y \leftarrow D(0)$ and outputs a constant-program that always outputs y.

- This achieves ideal one-time security: $\mathsf{OTP}^*(D)$ is simulatable by one query.
- But this construction is one-time correct only for constant-distribution functionalities.

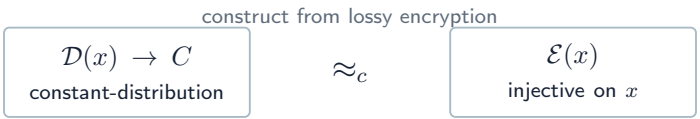
Insight: Whether a program samples from the same distribution for every input should not be efficiently learnable. (Use crypto to formalize this.)

Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.

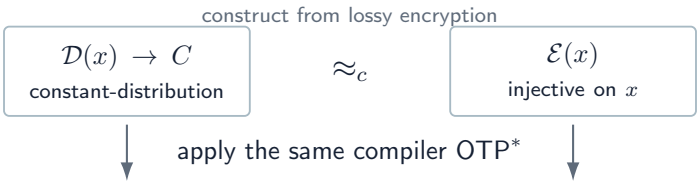
Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.



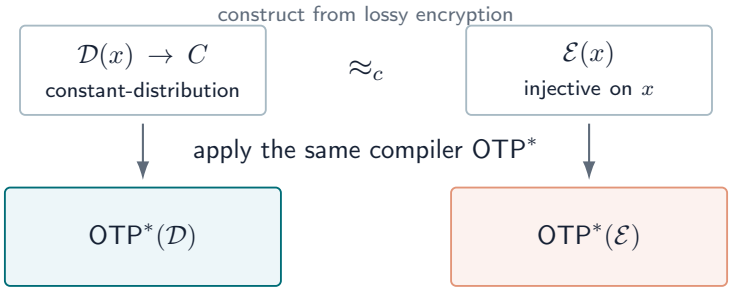
Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP*.



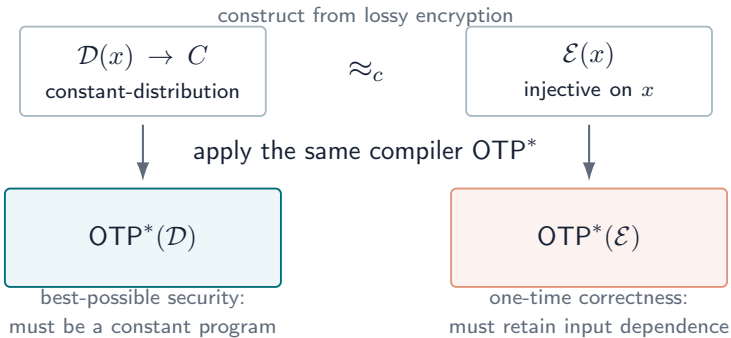
Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.



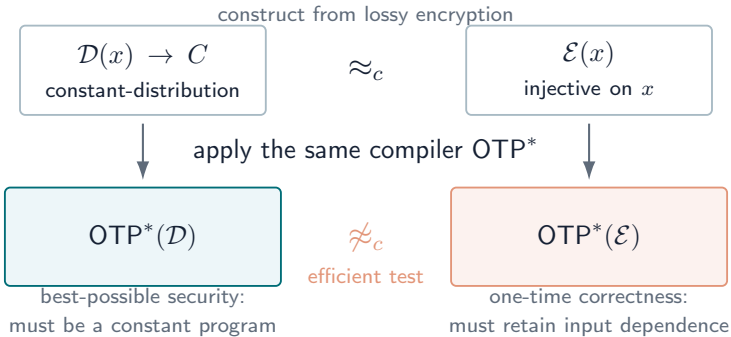
Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.



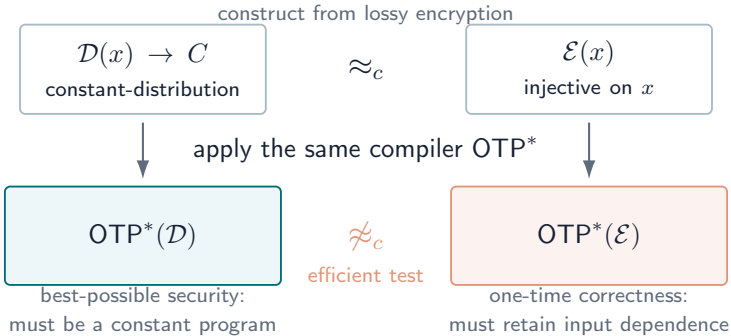
Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.



Formalizing the contradiction

Suppose there is a generic best-possible one-time compiler OTP^*.



Contradiction: OTP^* must be inefficient.

Why $\mathsf{OTP}^*(\mathcal{D})$ must be a constant program

Why $\mathsf{OTP}^*(\mathcal{D})$ must be a constant program

- A constant-distribution sampler D is one-time equivalent to a mixture of constant functions:

$$C_y(x; r) = y \text{ for } y \leftarrow D().$$

Why $\mathsf{OTP}^*(\mathcal{D})$ must be a constant program

- A constant-distribution sampler D is one-time equivalent to a mixture of constant functions:

$$C_y(x; r) = y \text{ for } y \leftarrow D().$$

- By best-possible security, $\mathsf{OTP}^*(D)$ must be simulatable from one sample of C_y.

Why $\mathsf{OTP}^*(\mathcal{D})$ must be a constant program

- A constant-distribution sampler D is one-time equivalent to a mixture of constant functions:
$$C_y(x; r) = y \text{ for } y \leftarrow D().$$
- By best-possible security, $\mathsf{OTP}^*(D)$ must be simulatable from one sample of C_y.
- By correctness, given only one sample of C_y, the simulator must output a constant program.

Getting around the best-possible impossibility

Two approaches:

- Restrict the class of protected functionalities from all randomized/quantum functionalities.
- Restrict the class of one-time compilers from all efficient implementations.

Getting around the best-possible impossibility, approach 2

Our definition (and proof) compares the given OTP to *any* implementation, including mixed implementations:

$$\rho = \sum_i p_i |\phi_i\rangle\langle\phi_i|.$$

We allow a program to be one-time correct only after averaging over p_i.

Getting around the best-possible impossibility, approach 2

Our definition (and proof) compares the given OTP to *any* implementation, including mixed implementations:

$$\rho = \sum_i p_i |\phi_i\rangle\langle\phi_i|.$$

We allow a program to be one-time correct only after averaging over p_i.

Question: What if we only focus on best possible within a restricted subclass of one-time compilers that output "pure programs"?

Naive pure BPOTP still allows random purification

The counterexample above

$$\rho = \sum_y p_y |C_y\rangle\langle C_y|$$

can be purified as

$$\rho = \mathbb{E}_r \left[|\psi_r\rangle\langle\psi_r| \right] \text{ where } |\psi_r\rangle = \sum_y \sqrt{p_y} e^{i r_y} |C_y\rangle.$$

$|\psi_r\rangle$ is pure and one-time correct for every r so the same impossibility still applies.

Naive pure BPOTP still allows random purification

The counterexample above

$$\rho = \sum_y p_y |C_y\rangle\langle C_y|$$

can be purified as

$$\rho = \mathbb{E}_r \left[|\psi_r\rangle\langle\psi_r| \right] \text{ where } |\psi_r\rangle = \sum_y \sqrt{p_y} e^{i r_y} |C_y\rangle.$$

$|\psi_r\rangle$ is pure and one-time correct for every r so the same impossibility still applies.

A pure-state OTP class needs a more careful definition.

Testable programs

Testable programs

Make the designated pure state operationally visible by including its reflection:

$$(|\psi\rangle, R_\psi) \qquad R_\psi = I - 2|\psi\rangle\langle\psi|.$$

Testable programs

Make the designated pure state operationally visible by including its reflection:

$$(|\psi\rangle, R_\psi) \qquad R_\psi = I - 2|\psi\rangle\langle\psi|.$$

R_ψ *tests* for the state $|\psi\rangle$.

Testable programs

Make the designated pure state operationally visible by including its reflection:

$$(|\psi\rangle, R_\psi) \qquad R_\psi = I - 2|\psi\rangle\langle\psi|.$$

R_ψ *tests* for the state $|\psi\rangle$.

Changing the hidden phases also affects R_ψ, not just its mathematical decomposition.

Testable programs

Make the designated pure state operationally visible by including its reflection:

$$(|\psi\rangle, R_\psi) \qquad R_\psi = I - 2|\psi\rangle\langle\psi|.$$

R_ψ *tests* for the state $|\psi\rangle$.

> Changing the hidden phases also affects R_ψ, not just its mathematical decomposition.

Definition: A *testable one-time compiler* is a one-time compiler that outputs mixtures over $(|\psi\rangle, R_\psi)$ where every $|\psi\rangle$ is a pure one-time correct implementation.

Testability of known OTP constructions

OTP construction	**Testable?**
Classical program	✅ Reflecting for computational basis is trivial
Uncloneable $\|\text{token}\rangle$ + obfuscated program that only computes P only with a valid $\|\text{token}\rangle$	✅ Use the obfuscated program
Mixures of constant programs for constant distributions	🟥 Mixed implementation: intentionally excluded

Testable OTPs capture all known OTP constructions in prior works.

Theorem 2: best-possible testable OTP

Best-possible *testable* one-time compiler exists for all randomized functionalities relative to a classical oracle.

- Also extends to all quantum functionalities $\downarrow$

A SEQ detour

Classical Single Effective Query (CSEQ): one-time simulation security introduced by Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25

A SEQ detour

Classical Single Effective Query (CSEQ): one-time simulation security introduced by Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25

- For a randomized functionality f, the view from querying the OTP is simulatable through a restricted CSEQ interface:

$$A(\mathsf{OTP}(f)) \approx_c \mathsf{Sim}^{O_f^{\mathsf{CSEQ}}}.$$

A SEQ detour

Classical Single Effective Query (CSEQ): one-time simulation security introduced by Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25

- For a randomized functionality f, the view from querying the OTP is simulatable through a restricted CSEQ interface:

$$A(\mathsf{OTP}(f)) \approx_c \mathsf{Sim}^{O_f^{\mathsf{CSEQ}}}.$$

- **High-level idea:** CSEQ interface allows for one query but refuses to collaborate as much as possible otherwise. Example: Compute/uncompute is allowed; but once the output is meaningfully disturbed, subsequent queries should stop working.

A SEQ detour

Classical Single Effective Query (CSEQ): one-time simulation security introduced by Gupte–Liu–Raizes–Roberts–Vaikuntanathan'25

- For a randomized functionality f, the view from querying the OTP is simulatable through a restricted CSEQ interface:

$$A(\mathsf{OTP}(f)) \approx_c \mathsf{Sim}^{O_f^{\mathsf{CSEQ}}}.$$

- **High-level idea:** CSEQ interface allows for one query but refuses to collaborate as much as possible otherwise. Example: Compute/uncompute is allowed; but once the output is meaningfully disturbed, subsequent queries should stop working.

- **Known:** CSEQ-secure OTPs for all randomized classical functionalities in the classical oracle model.

The CSEQ interface

Definition (The Classical Single Effective Query Oracle). For a randomized function $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$, we define the classical single effective query oracle O_f^{CSEQ} as implementing the following algorithm:

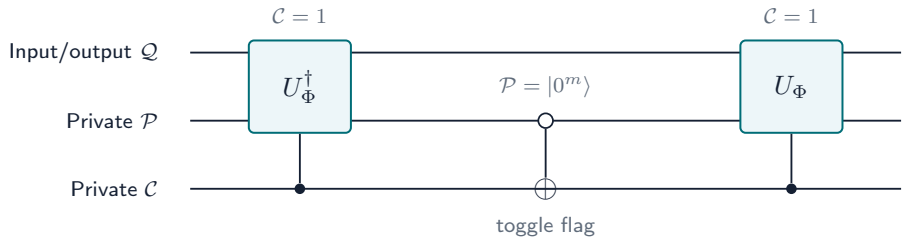
- We assume it has oracle access to O_f, which maps
$$|x, r, u\rangle \mapsto |x, r, u \oplus f(x; r)\rangle.$$
- The oracle maintains two internal registers: a workspace register $\mathcal{W}$ to perform intermediate computations, and a database register $\mathcal{D}$ to implement the database for the compressed random oracle. These registers are initialized to $|0\rangle_{\mathcal{W}} \otimes |\emptyset\rangle_{\mathcal{D}}$.
- To describe its behavior on queries, all we need to do is describe its behavior on basis states. We will maintain the invariant that the workspace register is always $|0\rangle_{\mathcal{W}}$.
- On query $|x, u, b\rangle_{\mathcal{Q}}$ on query register $\mathcal{Q}$, the oracle implements an isometry specified by the following steps on each basis state
$$|x, u, b\rangle_{\mathcal{Q}} \otimes |D\rangle_{\mathcal{D}} \otimes |0\rangle_{\mathcal{W}}.$$

1. If there exists some $x' \neq x$ such that $(x', r) \in D$ for some $r \in \mathcal{R}$, skip the following steps.
2. Copy x into the workspace register
$$|x, u, b\rangle_{\mathcal{Q}}|D\rangle_{\mathcal{D}}|0\rangle_{\mathcal{W}} \mapsto |x, u, b\rangle_{\mathcal{Q}}|D\rangle_{\mathcal{D}}|x, 0\rangle_{\mathcal{W}}.$$
3. Apply a compressed random oracle query isometry with query register $\mathcal{W}$ and database register $\mathcal{D}$.
4. On the basis states apply the following unitary map that acts on registers $\mathcal{Q}, \mathcal{W}$:
$$|x, u, b\rangle_{\mathcal{Q}}|D\rangle_{\mathcal{D}}|x, r\rangle_{\mathcal{W}} \mapsto |x, u \oplus f(x; r), b \oplus 1\rangle_{\mathcal{Q}}|D\rangle_{\mathcal{D}}|x, r\rangle_{\mathcal{W}}.$$
5. Uncompute the workspace register: first, query the compressed oracle again with the query register being $\mathcal{W}$ and the database register being $\mathcal{D}$, and then copy the input x.

(Generalized) SEQ interface for quantum functionalities

Definition (The Single Effective Query Oracle). Let Φ be a channel with any Stinespring unitary $U_\Phi : \mathcal{H}_\mathcal{Q} \otimes \mathcal{H}_\mathcal{P} \to \mathcal{H}_\mathcal{Q} \otimes \mathcal{H}_\mathcal{P}$, with $\mathcal{P}$ initialized to $|0^m\rangle_\mathcal{P}$. The oracle O_Φ^{SEQ} acts on the query register $\mathcal{Q}$ and maintains hidden registers $\mathcal{P}$ and $\mathcal{C}$, initialized to $|0^m\rangle_\mathcal{P}|0\rangle_\mathcal{C}$:

$$|\varphi\rangle_\mathcal{Q}|0^m\rangle_\mathcal{P}|0\rangle_\mathcal{C} \;\longleftrightarrow\; U_\Phi(|\varphi\rangle_\mathcal{Q}|0^m\rangle_\mathcal{P})\,|1\rangle_\mathcal{C}.$$



Example: SEQ on random-coin channel

Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r \in \{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Example: SEQ on random-coin channel

Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r\in\{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Consider Stinespring unitary

$$U_{\Phi_n} : |\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}} \longmapsto 2^{-n/2} \sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}.$$

Example: SEQ on random-coin channel

Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r\in\{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Consider Stinespring unitary

$$U_{\Phi_n} : |\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}} \longmapsto 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}.$$

Then $O^{\mathsf{SEQ}}_{\Phi_n}$ hides $\mathcal{P}, \mathcal{C}$ and swaps

$$|\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}}|0\rangle_{\mathcal{C}} \longleftrightarrow 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}|1\rangle_{\mathcal{C}}.$$

Example: SEQ on random-coin channel

Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r\in\{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Consider Stinespring unitary

$$U_{\Phi_n} : |\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}} \longmapsto 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}.$$

Then $O^{\text{SEQ}}_{\Phi_n}$ hides $\mathcal{P},\mathcal{C}$ and swaps

$$|\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}}|0\rangle_{\mathcal{C}} \longleftrightarrow 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}|1\rangle_{\mathcal{C}}.$$

Example: SEQ on random-coin channel

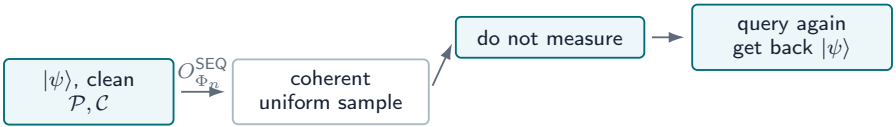
Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r\in\{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Consider Stinespring unitary

$$U_{\Phi_n} : |\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}} \longmapsto 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}.$$

Then $O^{\text{SEQ}}_{\Phi_n}$ hides $\mathcal{P},\mathcal{C}$ and swaps

$$|\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}}|0\rangle_{\mathcal{C}} \longleftrightarrow 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}|1\rangle_{\mathcal{C}}.$$



Example: SEQ on random-coin channel

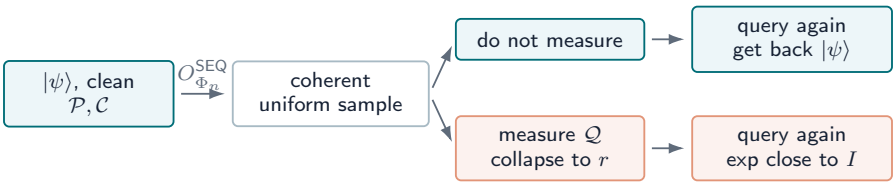
Channel: $\Phi_n(\rho_{\mathcal{Q}}) = 2^{-n} \sum_{r\in\{0,1\}^n} |r\rangle\langle r|_{\mathcal{Q}}$ discards the input.

Consider Stinespring unitary

$$U_{\Phi_n} : |\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}} \longmapsto 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}.$$

Then $O^{\text{SEQ}}_{\Phi_n}$ hides $\mathcal{P},\mathcal{C}$ and swaps

$$|\psi\rangle_{\mathcal{Q}}|0^n\rangle_{\mathcal{P}_R}|0^n\rangle_{\mathcal{P}_{\text{in}}}|0\rangle_{\mathcal{C}} \longleftrightarrow 2^{-n/2}\sum_r |r\rangle_{\mathcal{Q}}|r\rangle_{\mathcal{P}_R}|\psi\rangle_{\mathcal{P}_{\text{in}}}|1\rangle_{\mathcal{C}}.$$



Theorem 3: SEQ subsumes CSEQ

For every non-trivial randomized classical $f : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ with $|\mathcal{X}| > 1$, there is a channel Φ_f such that O_f^{CSEQ} and $O_{\Phi_f}^{\mathsf{SEQ}}$ are efficiently inter-simulatable.

Thus SEQ generalizes and simplifies CSEQ.

CSEQ = SEQ using channel wrapper Φ_f.

The proof is involved because CSEQ is involved.

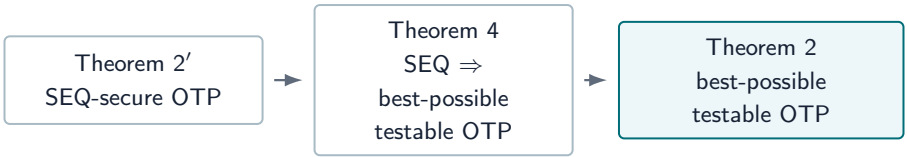
Theorem 4: SEQ $\Rightarrow$ best-possible testable OTPs

Any SEQ-secure one-time compiler is best-possible among testable one-time compilers.
Moreover, this implication relativizes to all unitary oracles.

SEQ plays for testable OTPs the role that VBB plays for obfuscation.

Theorem 2′: SEQ-secure OTP

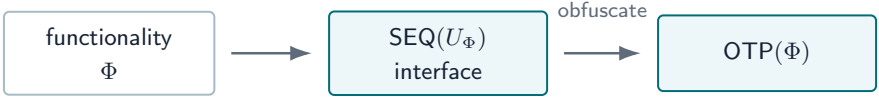
There exists an SEQ-secure one-time compiler for all quantum functionalities in the classical oracle model.



Combining Theorem 2′ with Theorem 4 gives the original Theorem 2.

Construction overview

Idea: it suffices to obfuscate the SEQ interface.



The catch: SEQ has private states

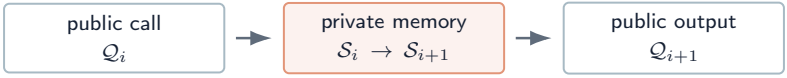
Not a stateless unitary $\mathcal{Q} \to \mathcal{Q}$

New primitive: stateful iO

A stateful program is a unitary plus an initial private state.

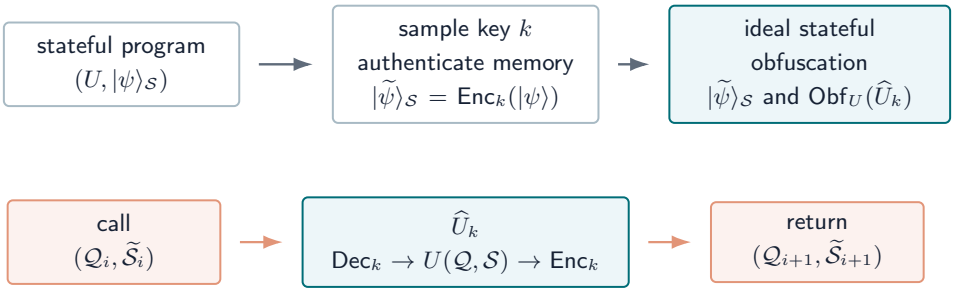
indistinguishable behavior for every polynomial sequence of calls

$\Longrightarrow$ indistinguishable obfuscations



Constructing ideal stateful obfuscation

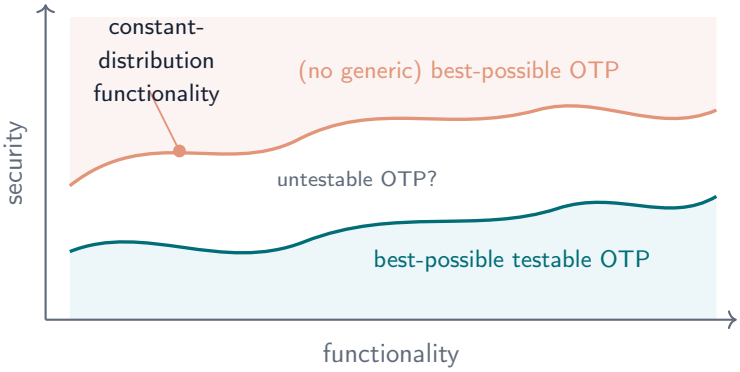
Protect the private memory, then obfuscate the memory-update unitary.



Ideal unitary obfuscation can be instantiated in the classical oracle model [Huang–Tang'25].

Open: replace ideal obfuscation with (quantum) iO

Conclusions



See paper for a generic multi-observable attack against testable OTPs.