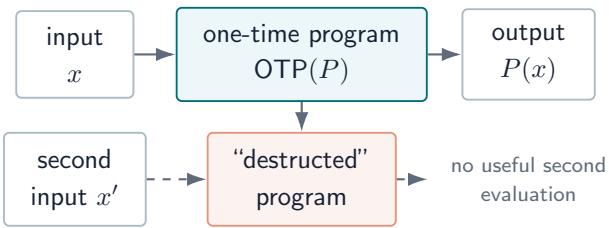
On Best-Possible One-Time Programs

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

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

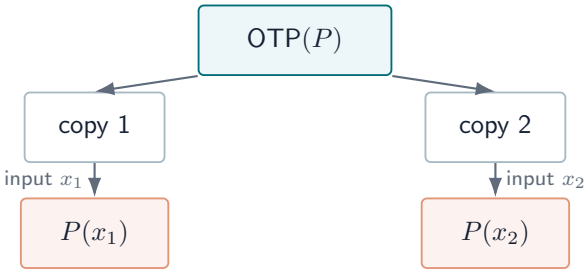
Ideal goal

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.



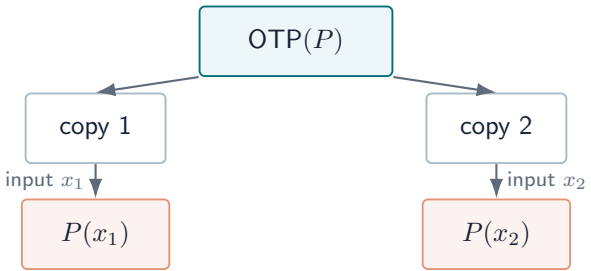
Classical OTPs are impossible

As long as OTP outputs classical bitstrings...



Classical OTPs are impossible

As long as OTP outputs classical bitstrings...



Clearly a one-time program cannot be solely (classical) software based, as (classical) software can always be copied and run again, enabling a user to execute the program more times than specified. —GKR'08

Quantum????

one-time programs should only allow for 1 evaluation

quantum states only allow for 1 complete measurement

Quantum????

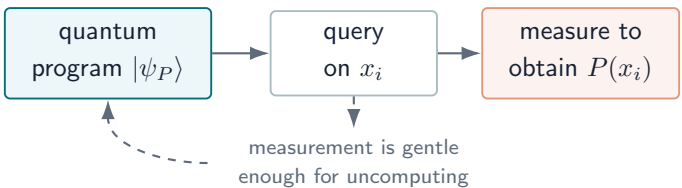
one-time programs should only allow for 1 evaluation

quantum states only allow for 1 complete measurement

Quantum OTPs are also impossible

Broadbent–Gutoski–Stebila'13 impossibility

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

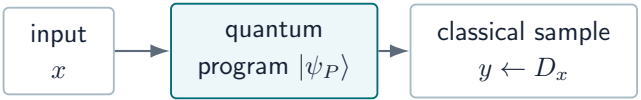


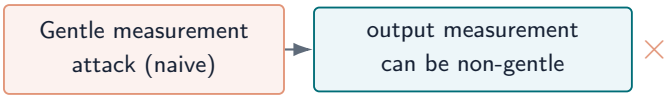
A second attack: coherent access

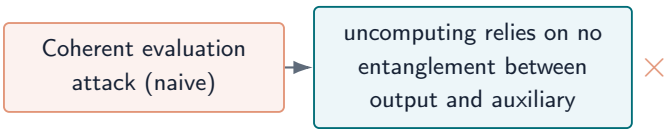
Coherent computation is always possible even if implemented by a quantum state:

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

A way out? Quantum program + randomized outputs



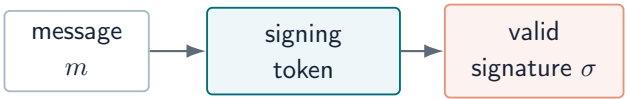




Signature tokens [Ben-David–Sattath'19]

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

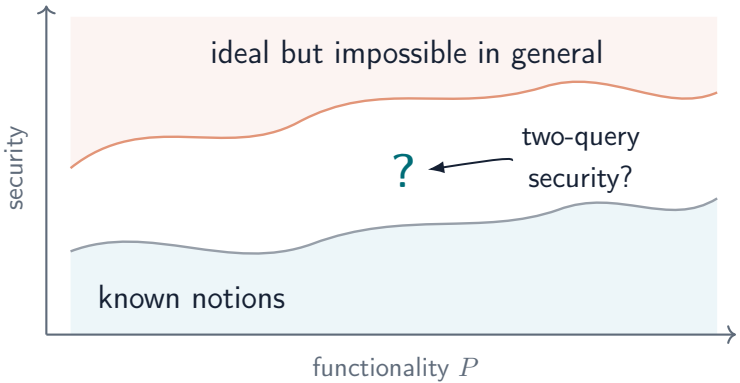
For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

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)

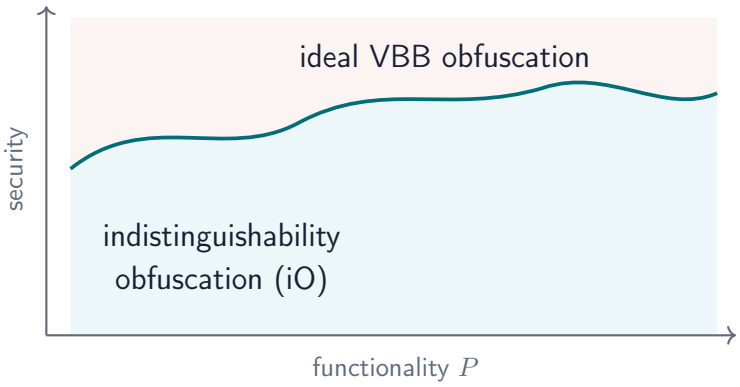
For simplicity, we use ideal classical obfuscation / classical oracles as assumptions.

What is the strongest achievable security?

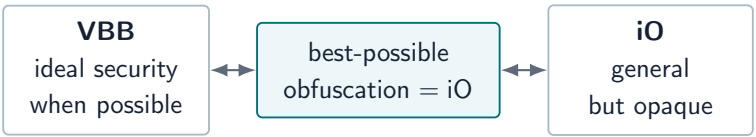


What's the least amount of information that can be revealed while allowing for a single evaluation?

Many-time analogy: obfuscation



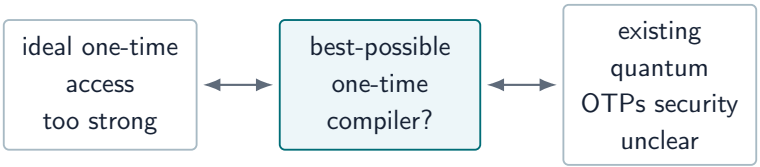
Best-Possible Obfuscation [Goldwasser–Rothblum'07]



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 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

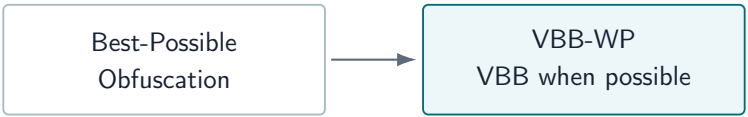
- Background and motivation
  (you are here)
- Best-possible one-time programs are impossible
  - Proof sketch
- Getting around the impossibility

One-Time VBB

A one-time compiler OTP achieves **one-time VBB** for functionality $\mathcal{P}$, if for any adversary A, there exists a one-query simulator Sim such that

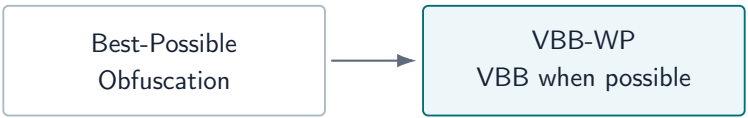
$$\mathsf{A}(\mathsf{OTP}(P)) \approx_c \mathsf{Sim}^P(1^{|P|}).$$

VBB When Possible



VBB-WP: Achieves VBB if there exists a VBB obfuscator for the functionality. Otherwise, no security guarantees.

VBB When Possible



VBB-WP: Achieves VBB if there exists a VBB obfuscator for the functionality. Otherwise, no security guarantees.

Similarly define one-time VBB-WP, and best-possible OTP implies one-time VBB-WP.

Theorem 1: no best-possible one-time programs

Assuming lossy encryptions exist,
no generic one-time compiler can satisfy one-time VBB-WP for randomized functionalities.

- Corollary: No best-possible OTPs either.
- Lossy encryptions can be constructed from Learning with Errors (LWE) or from weak pseudorandom group actions.
- Applies to both quantum and classical; does NOT apply to NISQ-style adversaries nor one-time hardware.
- Also relativizes to all oracles.

First step: a trivial randomized functionality

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

First step: a trivial randomized functionality

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

Claim: There is a one-time VBB compiler for D.

First step: a trivial randomized functionality

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

Claim: There is a one-time VBB compiler for D.

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

First step: a trivial randomized functionality

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

Claim: There is a one-time VBB compiler for D.

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

- This achieves ideal one-time VBB: $\mathsf{OTP}^*(D)$ is simulatable by querying $D(0)$.

First step: a trivial randomized functionality

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

Claim: There is a one-time VBB compiler for D.

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

- This achieves ideal one-time VBB: $\mathrm{OTP}^*(D)$ is simulatable by querying $D(0)$.
- Observe: this construction is one-time correct only for constant-distribution functionalities.

First step: a trivial randomized functionality

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

Claim: There is a one-time VBB compiler for D.

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

- This achieves ideal one-time VBB: $\mathsf{OTP}^*(D)$ is simulatable by querying $D(0)$.
- Observe: this construction is one-time correct only for constant-distribution functionalities.

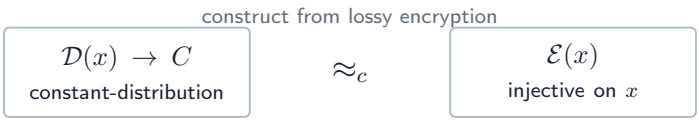
Insight: Whether a program samples from the same distribution for every input should not be efficiently learnable. (Use crypto to formalize unlearnability.)

Formalizing the contradiction

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

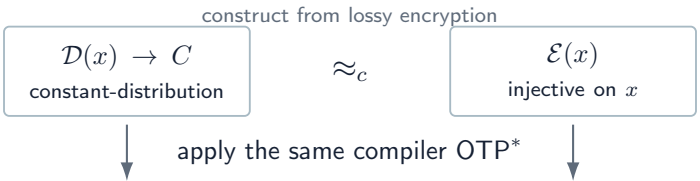
Formalizing the contradiction

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



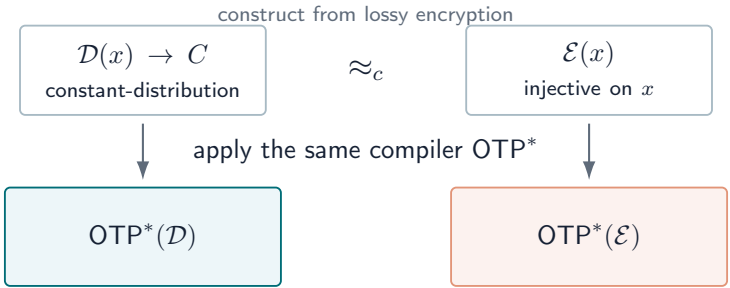
Formalizing the contradiction

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



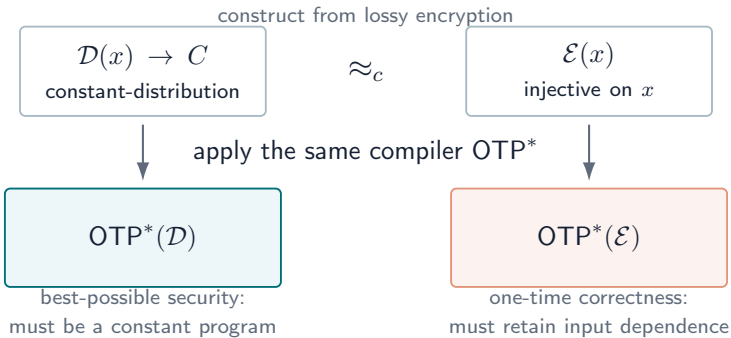
Formalizing the contradiction

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



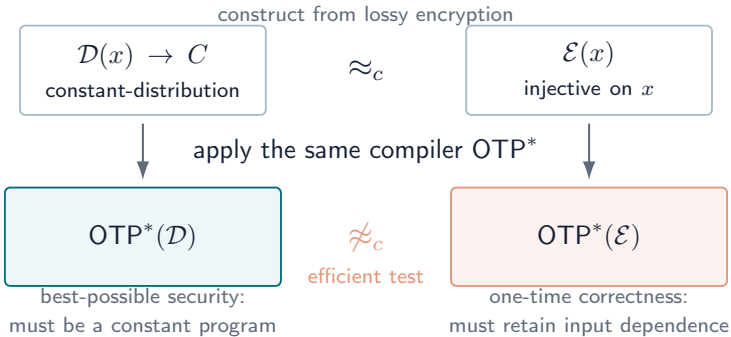
Formalizing the contradiction

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



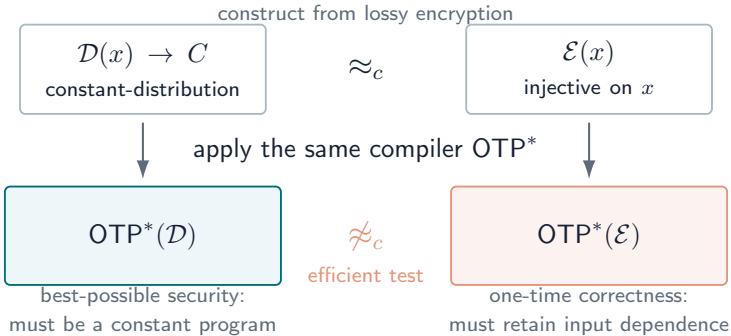
Formalizing the contradiction

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.

Getting around the best-possible impossibility

Two approaches:

- Restrict the class of supported 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 consider one-time compilers where every program in the support is one-time correct on its own?

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 $|\psi\rangle$: accepts $|\psi\rangle$ and rejects any other orthogonal state.

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 $|\psi\rangle$: accepts $|\psi\rangle$ and rejects any other orthogonal state.

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 $|\psi\rangle$: accepts $|\psi\rangle$ and rejects any other orthogonal state.

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.

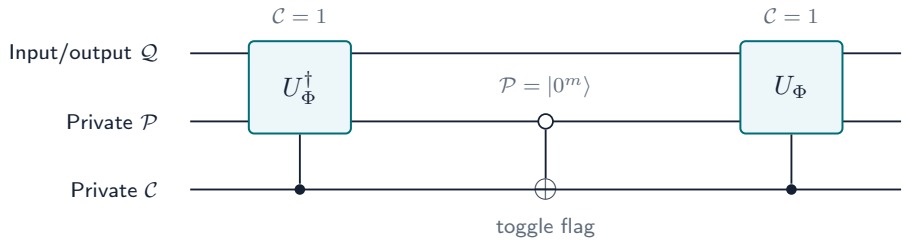
- Also extends to all quantum functionalities ↓

(Generalized) SEQ interface for quantum functionalities

The simulator instead of only being able to query the functionality once, is allowed to query the following restricted interface:

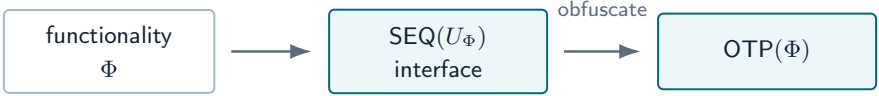
Definition (The Single Effective Query Oracle). Let Φ be a channel with any Stinespring unitary $U_\Phi : \mathcal{H}_\mathcal{Q} \otimes \mathcal{H}_\mathcal{P} \to \mathcal{H}_\mathcal{Q} \otimes \mathcal{H}_\mathcal{P}$, with $\mathcal{P}$ initialized to $|0^m\rangle_\mathcal{P}$. The oracle O_Φ^{SEQ}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

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



Construction overview

Lemma: Obfuscating the SEQ interface suffices for best-possible testable OTPs.

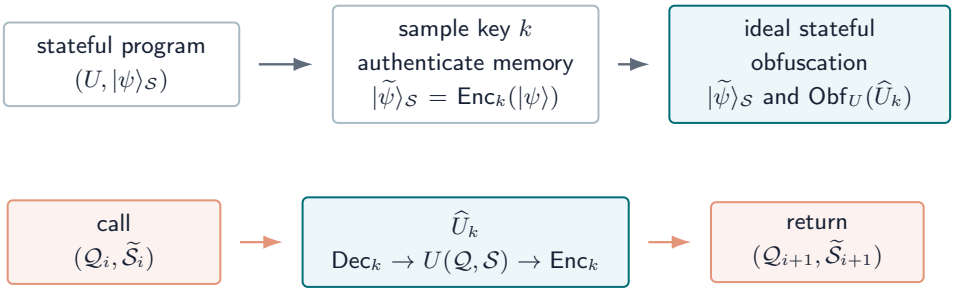


The catch: SEQ has private states

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

Constructing ideal stateful obfuscation

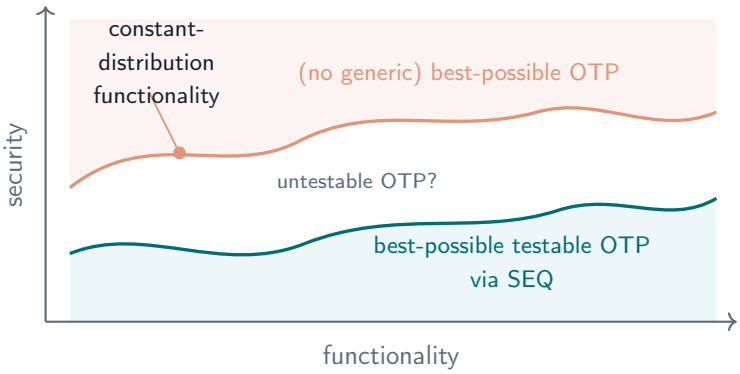
Protect the private memory, then obfuscate the memory-update unitary.



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.