Impossibility of VDFs in the ROM

The Complete Picture

Max von Consbruch*

Joint work with Hamza Abusalah†, Karen Azari*, Chethan Kamath‡ and Erkan Tairi§

*Uni Vienna
†IMDEA Madrid
‡IIT Bombay
§UC Berkeley

Overview

Main Theorem. Verifiable delay functions do not exist in the random oracle model.

Overview

Main Theorem. Verifiable delay functions do not exist in the random oracle model.

This Talk:

- Random Oracle Model

Overview

Main Theorem. Verifiable delay functions do not exist in the random oracle model.

This Talk:

- Random Oracle Model
- Verifiable Delay Functions in the Random Oracle Model

Overview

Main Theorem. Verifiable delay functions do not exist in the random oracle model.

This Talk:

- Random Oracle Model
- Verifiable Delay Functions in the Random Oracle Model
- Previous Results

Overview

Main Theorem. Verifiable delay functions do not exist in the random oracle model.

This Talk:

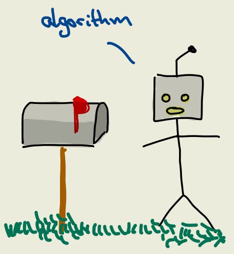
- Random Oracle Model
- Verifiable Delay Functions in the Random Oracle Model
- Previous Results
- Our Results

The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.

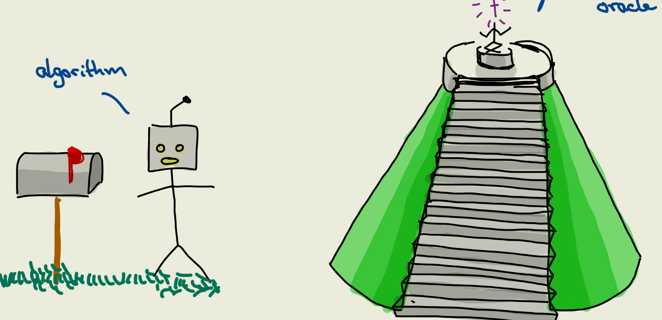
The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.



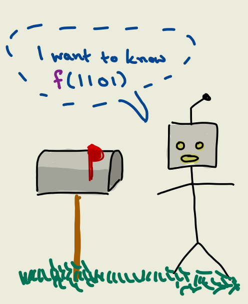
The (parallel, query-complexity) Random Oracle Model

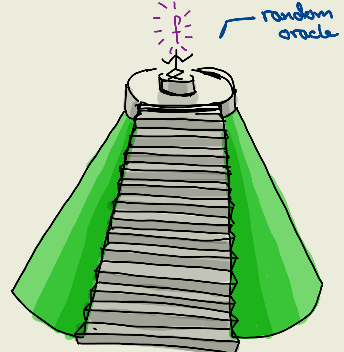
- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.



The (parallel, query-complexity) Random Oracle Model

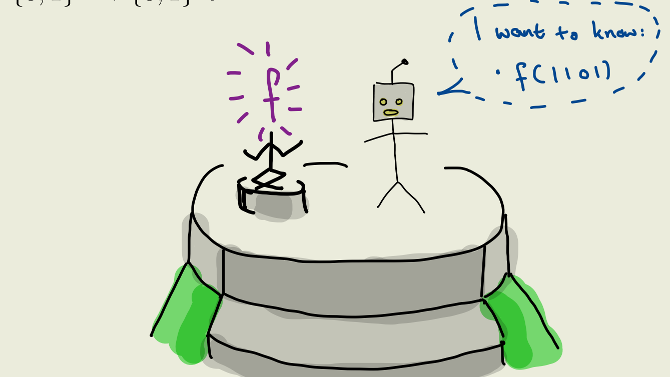
- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.





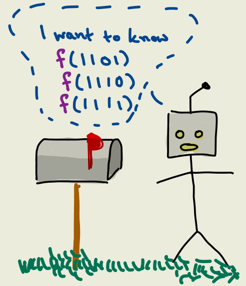
The (parallel, query-complexity) Random Oracle Model

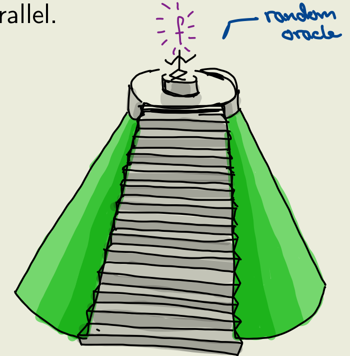
- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.



The (parallel, query-complexity) Random Oracle Model

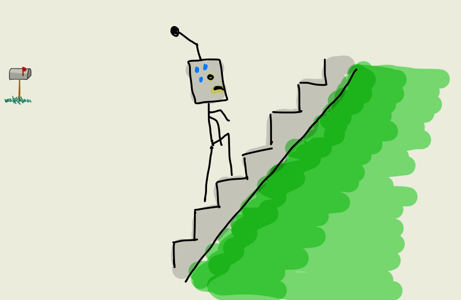
- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.





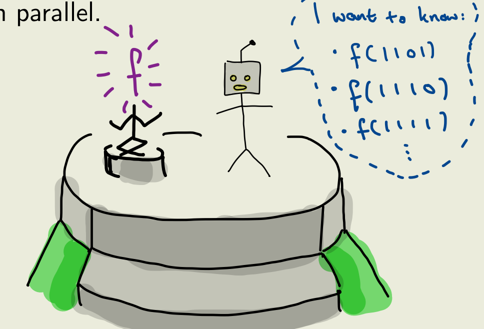
The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.



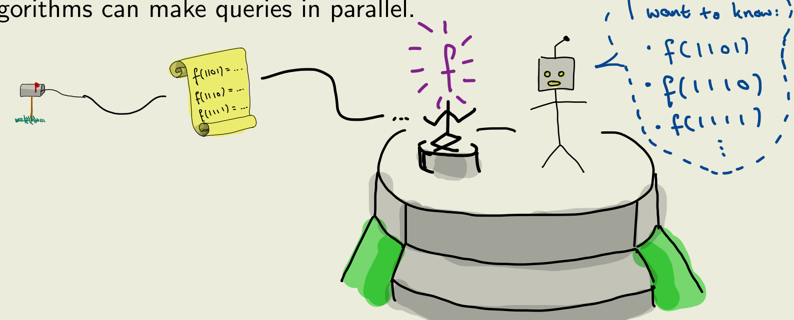
The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.



The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.

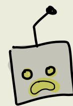


The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f: \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.

The (parallel, query-complexity) Random Oracle Model

- All algorithms have shared access to a random function "random oracle" $f\colon \{0,1\}^* \to \{0,1\}^\lambda$.
- Algorithms can make queries in parallel.
- Two measures for complexity:
    1. Query-Complexity ("how many queries in total?")
    2. Round-Complexity ("how many rounds of queries?")



ROM: Examples

Problem	Query Complexity	Round Complexity
Find $x \in \{0,1\}^*$ s.t. $f(x)$ has $\log_2(T)$ leading zeroes	T	1

$f(x) = \underbrace{00\cdots0}_{\log_2(T)}*\dots*$

ROM: Examples

Problem	**Query Complexity**	**Round Complexity**
Find $x \in \{0,1\}^*$ s.t. $f(x)$ has $\log_2(T)$ leading zeroes	T	1
Calculate $f^T(x)$ for given $x \leftarrow \{0,1\}^*$	T	T

$y = \underbrace{f(f(\ldots(f}_{T}(x))\ldots))$

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$

I want to know F(x)!

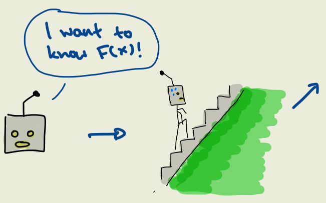
Example: $F(x) = f^T(x) = f(f(\dots(f(x))\dots))$.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$



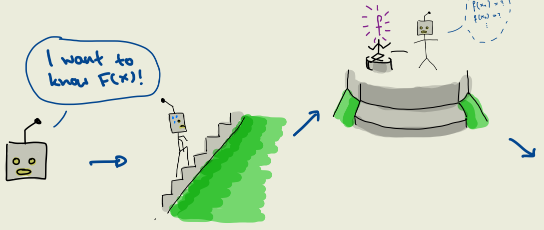
Example: $F(x) = f^T(x) = f(f(\dots(f(x))\dots))$.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$



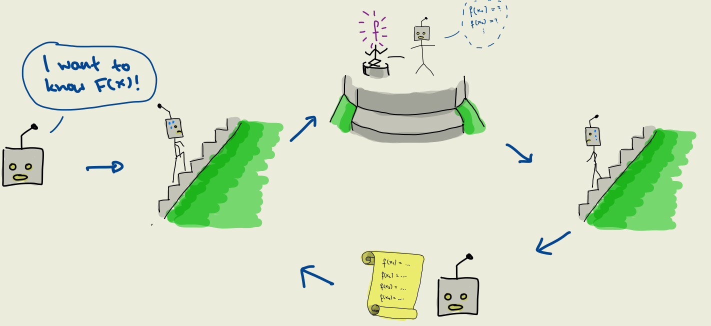
Example: $F(x) = f^T(x) = f(f(\ldots(f(x))\ldots))$.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$



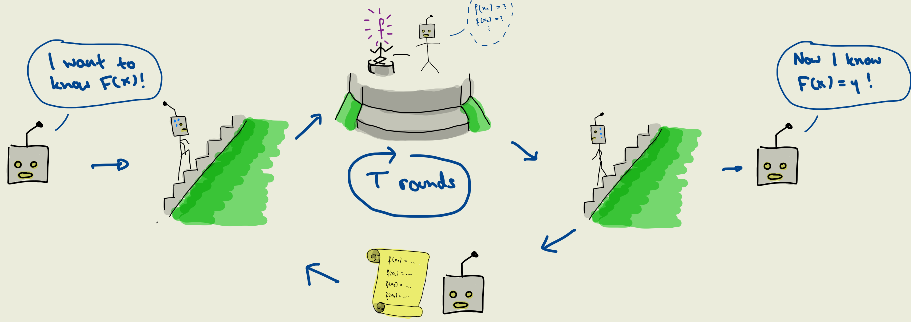
Example: $F(x) = f^T(x) = f(f(\ldots(f(x))\ldots))$.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$



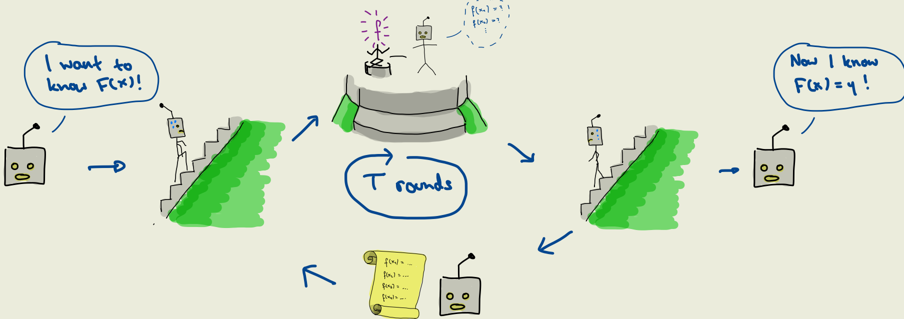
Example: $F(x) = f^T(x) = f(f(\ldots(f(x))\ldots)).$

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...

$$F : \mathcal{X} \to \mathcal{Y} \text{ can be evaluated in } T = \text{poly}(\lambda) \text{ rounds.}$$



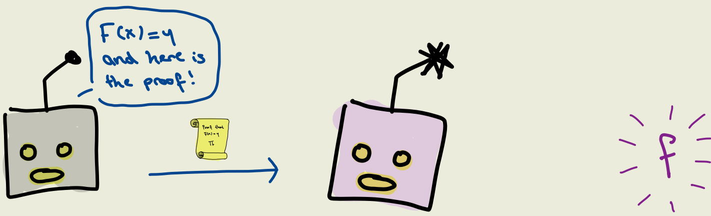
sequentiality: it is impossible to evaluate F in $\ll T$ rounds.

Example: $F(x) = f^T(x) = f(f(\dots(f(x))\dots))$.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...
- ...which is verifiable!

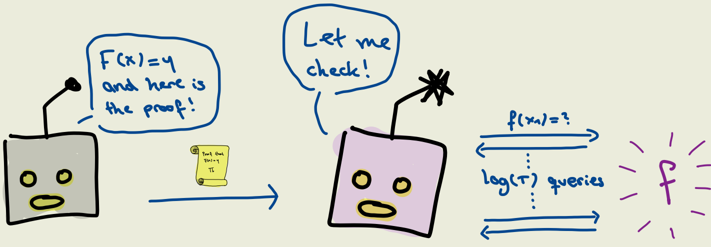


sequentiality: it is impossible to evaluate F in $\ll T$ rounds.

Verifiable Delay Functions in the ROM

A *Verifiable Delay Function (VDF)* is a...

- delay function...
- ...which is verifiable!



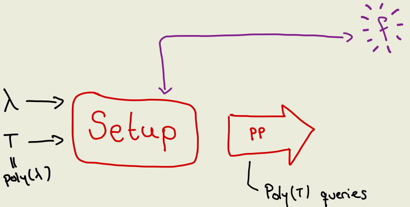
sequentiality: it is impossible to evaluate F in $\ll T$ rounds.

Verifiable Delay Functions in the ROM

A *VDF* is a triple of algorithms $(\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$

- a delay function...
- ...which is verifiable!

More precisely, it is a triple of algorithms with the following syntax.



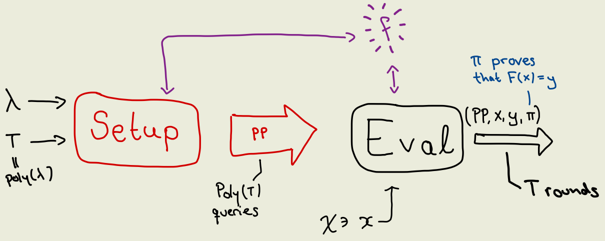
- **sequentiality:** it is impossible to evaluate F in $\ll T$ rounds.

Verifiable Delay Functions in the ROM

A *VDF* is a triple of algorithms $(\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$

- a delay function...
- ...which is verifiable!

More precisely, it is a triple of algorithms with the following syntax.



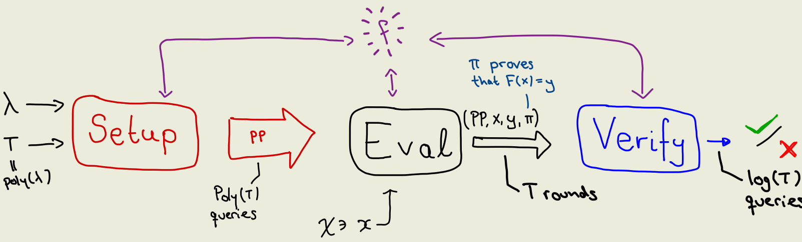
- **sequentiality:** it is impossible to evaluate F in $\ll T$ rounds.

Verifiable Delay Functions in the ROM

A *VDF* is a triple of algorithms $(\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$

- a delay function...
- ...which is verifiable!

More precisely, it is a triple of algorithms with the following syntax.



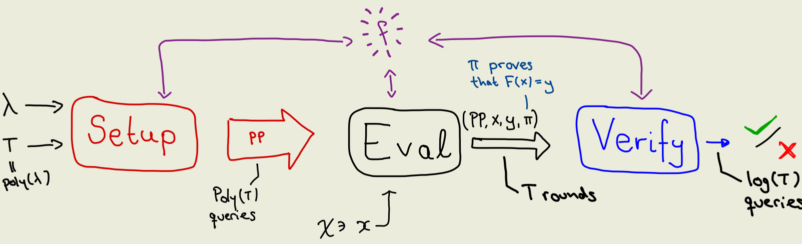
- **sequentiality:** it is impossible to evaluate F in $\ll T$ rounds.

Verifiable Delay Functions in the ROM

A *VDF* is a triple of algorithms $(\text{Setup}, \text{Eval}, \text{Verify})$

- a delay function...
- ...which is verifiable!

More precisely, it is a triple of algorithms with the following syntax.



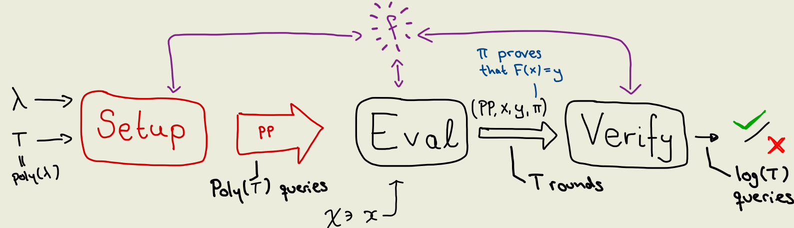
- **sequentiality:** it is impossible to evaluate F in $\ll T$ rounds.
- **soundness:** no polynomial-query algorithm can construct (x, y, π) with $y \neq F(x)$ and $\text{Verify}^f(\text{pp}, x, y, \pi) = 1$.

Breaking VDFs in the ROM

Assumption. $\Pi_{\mathrm{VDF}} = (\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$ is a *sound* VDF.

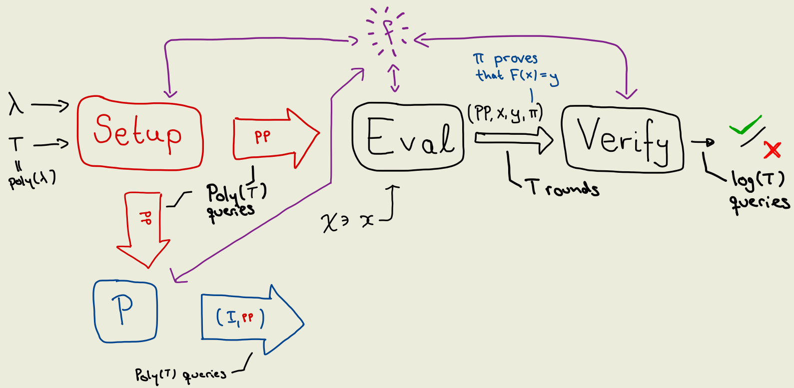
Breaking VDFs in the ROM

Assumption. $\Pi_{\text{VDF}} = (\text{Setup}, \text{Eval}, \text{Verify})$ is a *sound* VDF.
Aim. Show that Π_{VDF} does not satisfy *sequentiality.*



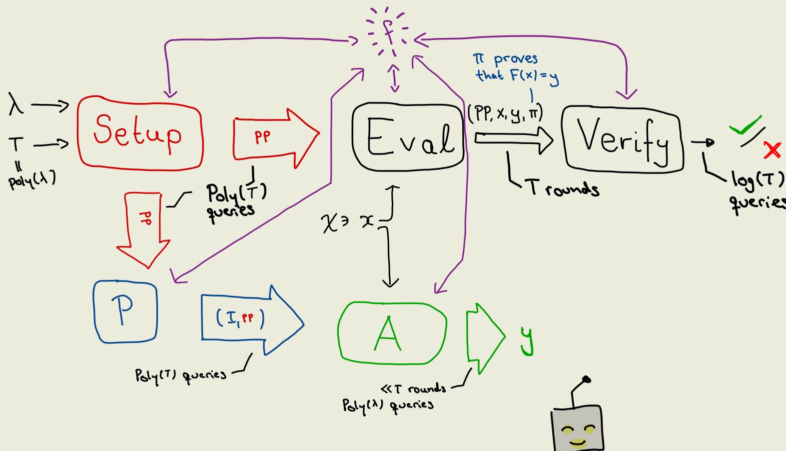
Breaking VDFs in the ROM

Assumption. $\Pi_{\mathrm{VDF}} = (\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$ is a *sound* VDF.
Aim. Show that Π_{VDF} does not satisfy *sequentiality.*



Breaking VDFs in the ROM

Assumption. $\Pi_{\mathrm{VDF}} = (\mathrm{Setup}, \mathrm{Eval}, \mathrm{Verify})$ is a *sound* VDF.
Aim. Show that Π_{VDF} does not satisfy *sequentiality.*



Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$

Overview of Previous Literature

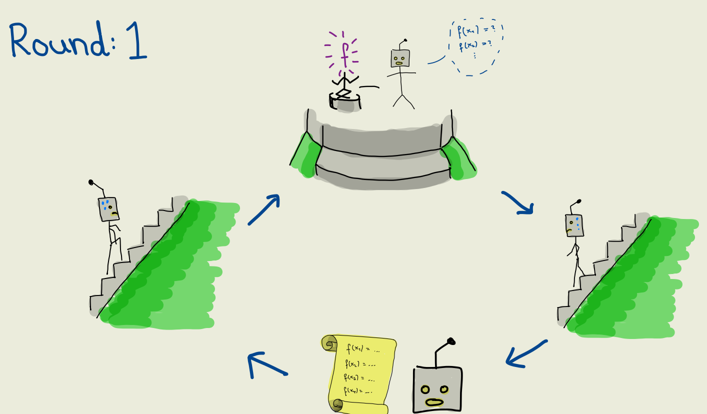
$$q = \#Q(\mathrm{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$
[GRY25]	deterministic	usual	$O(\log(T))$

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

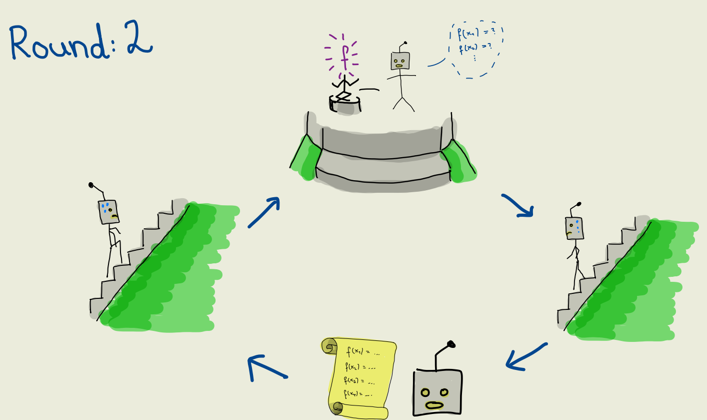
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking “tight” VDFs

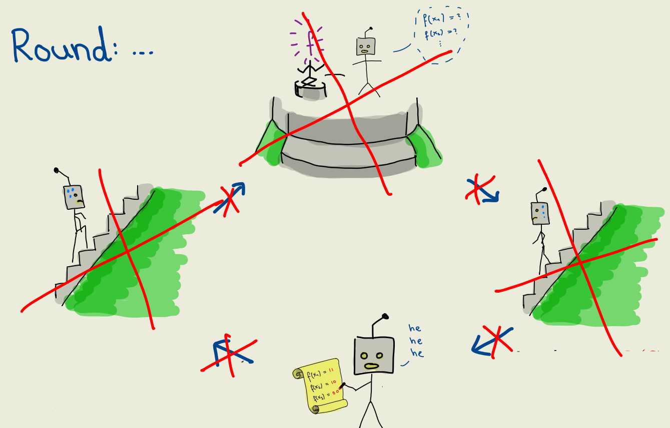
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

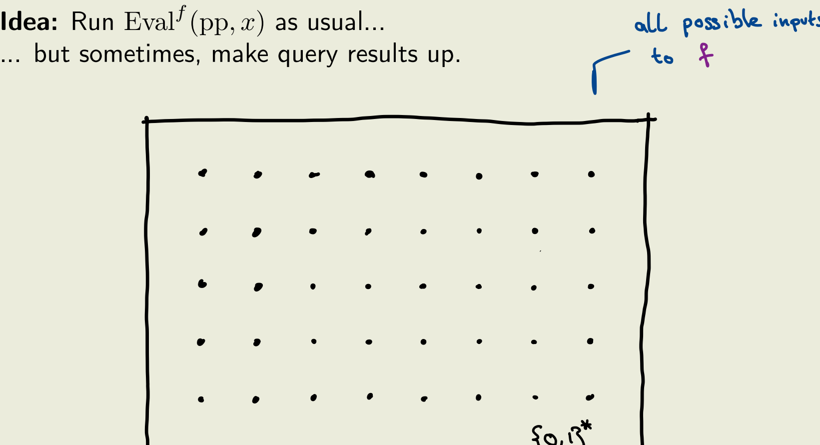
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

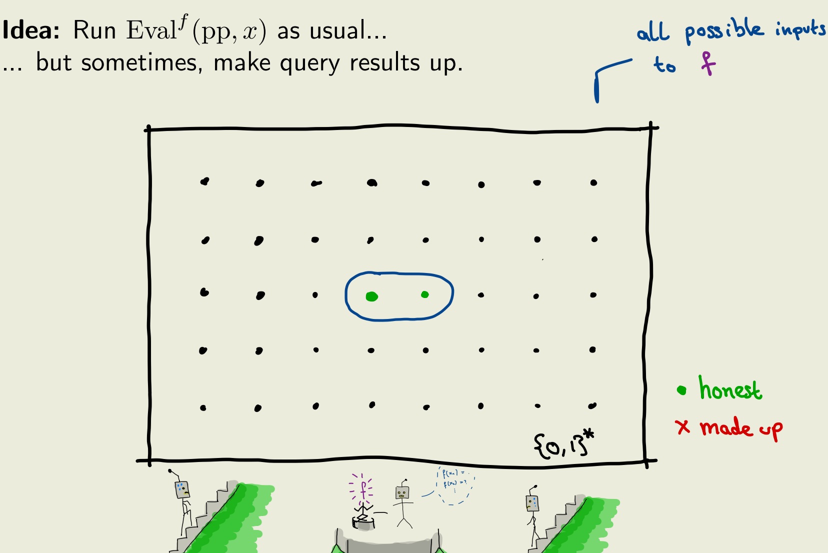
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

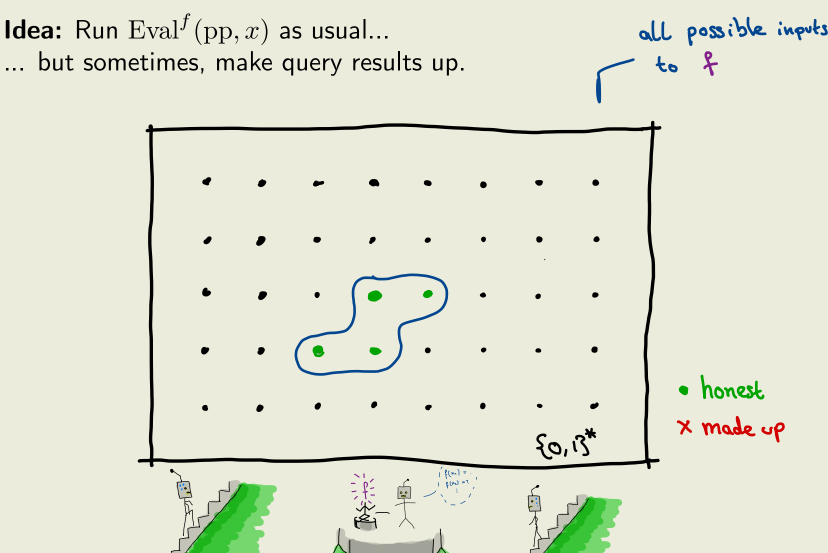
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

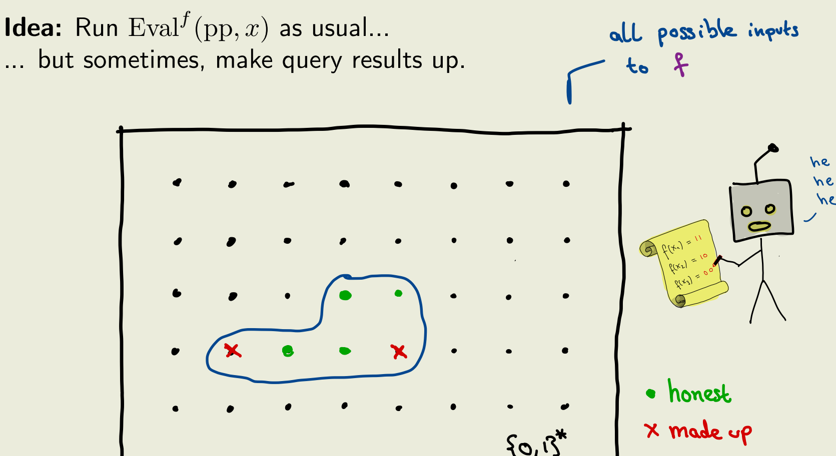
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

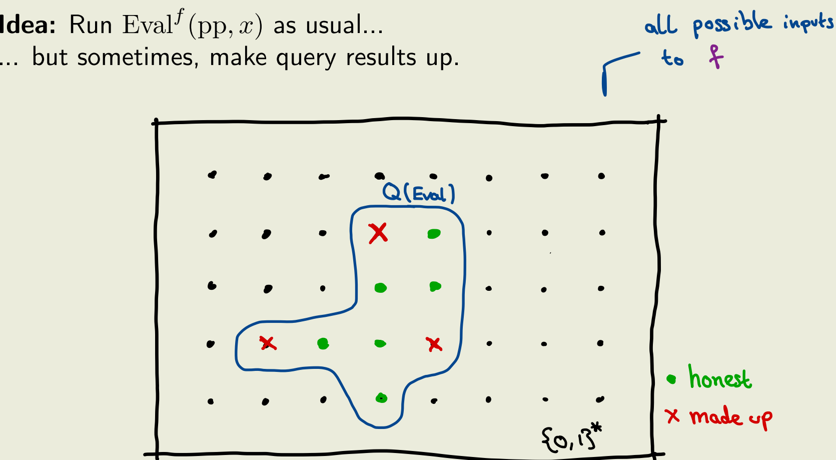
Breaking "tight" VDFs

Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator; [DGMV20]

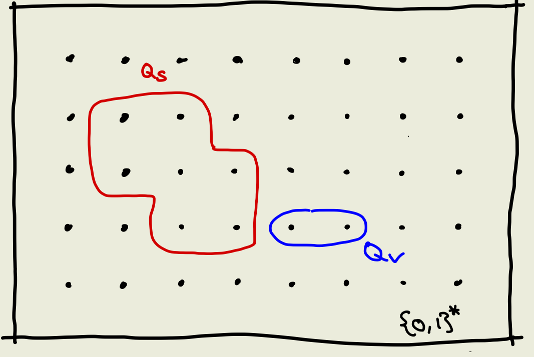
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

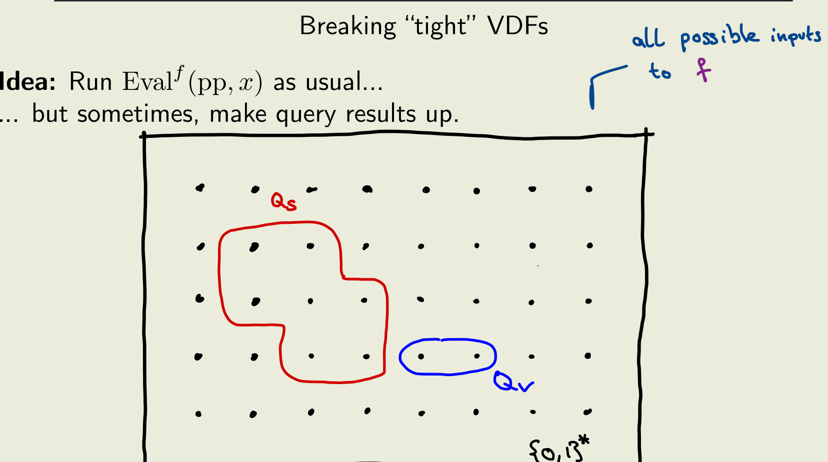
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



Main Observation: If none of the made up queries lie in $\color{red}{Q(\mathrm{Setup})}$ or $\color{blue}{Q(\mathrm{Verify})}$, then we win.

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking “tight” VDFs

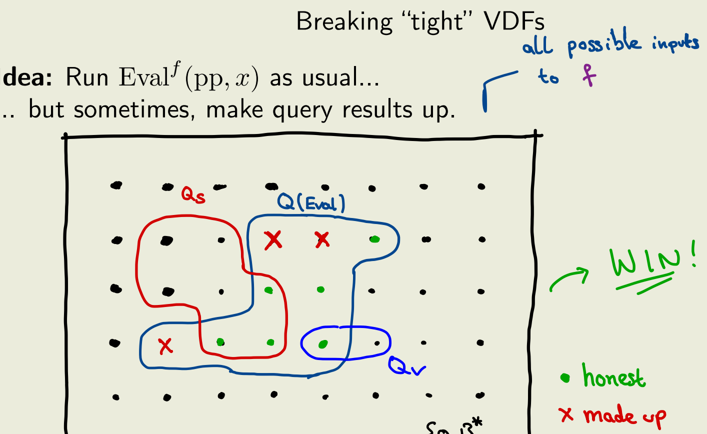
Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.

Main Observation: If none of the made up queries lie in $Q(\mathrm{Setup})$ or $Q(\mathrm{Verify})$, then we win.

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.



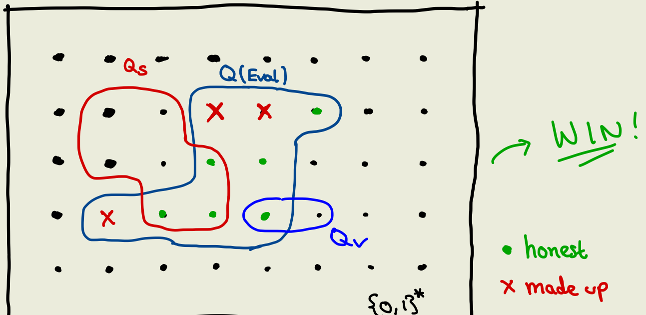
Main Observation: If none of the made up queries lie in $Q(\mathrm{Setup})$ or $Q(\mathrm{Verify})$, then we win.

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Idea: Run $\mathrm{Eval}^f(\mathrm{pp}, x)$ as usual...
... but sometimes, make query results up.

all possible inputs to f



Main Observation: If none of the made up queries lie in $Q(\mathrm{Setup})$ or $Q(\mathrm{Verify})$, then we win.
<u>Theorem.</u> With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.*

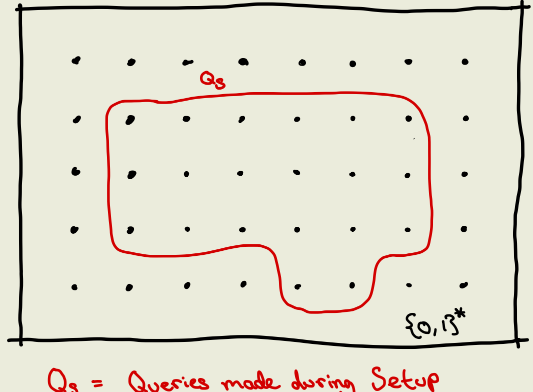
* with high probability

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?



Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!

Round: 1

Q_S

Q(Eval^f(x_1))

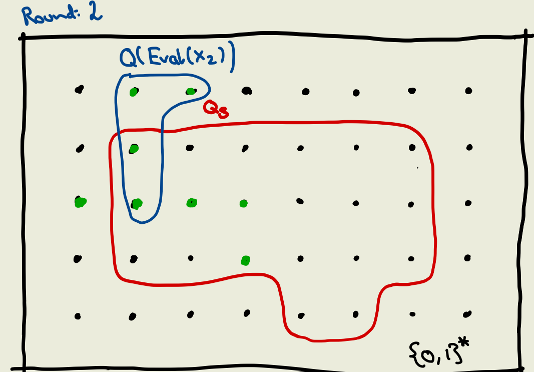
{0,1}*

Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!

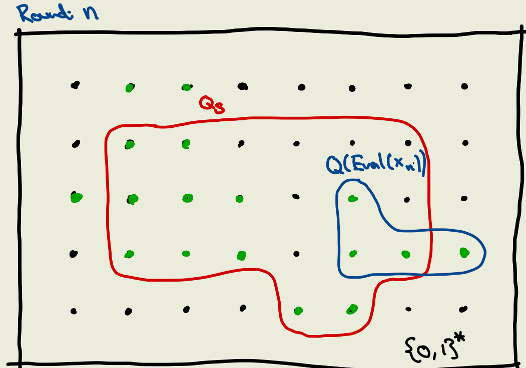


Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!

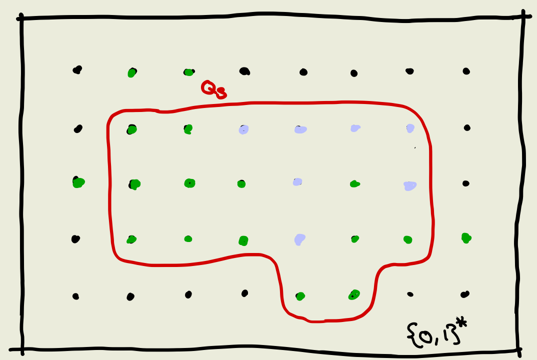


Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking “tight” VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!

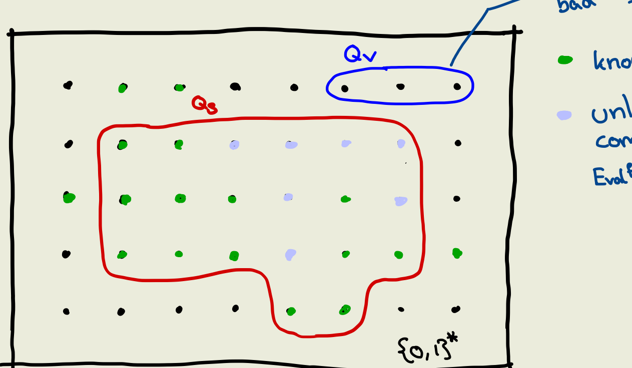


Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!

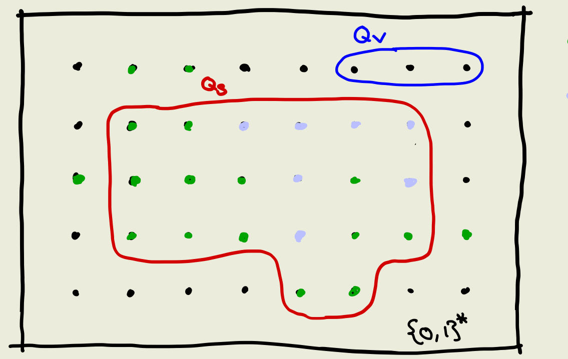


Attack 1: the Lazy Honest Evaluator [DGMV20]

Breaking "tight" VDFs

Theorem [DGMV20]. With $q = \#(\sout{Q(\text{Setup}) \cup}\, Q(\text{Verify}))$, we can evaluate F in $T - O(T/q)$ rounds, after preprocessing.

Problem: What if Setup makes a lot of queries?
Solution: use preprocessing to simulate *honest* Eval runs!



known

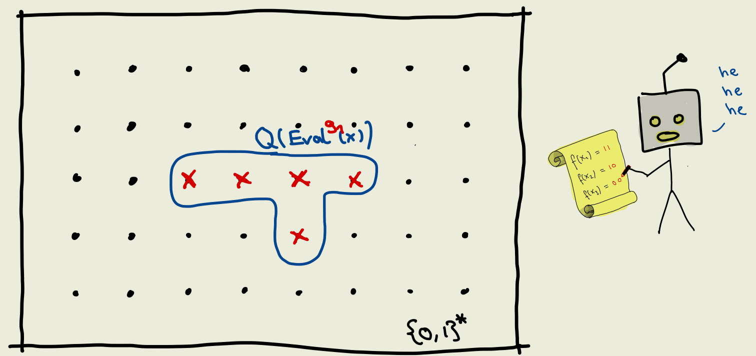
unlikely to come up in $\text{Eval}^f(x)$ for $x \leftarrow X$

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.

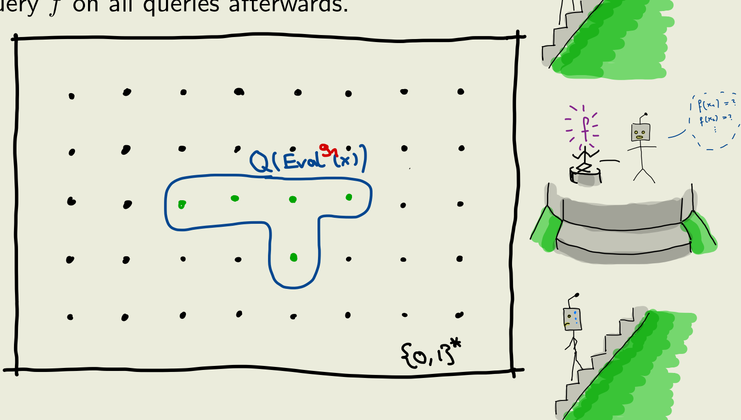


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.

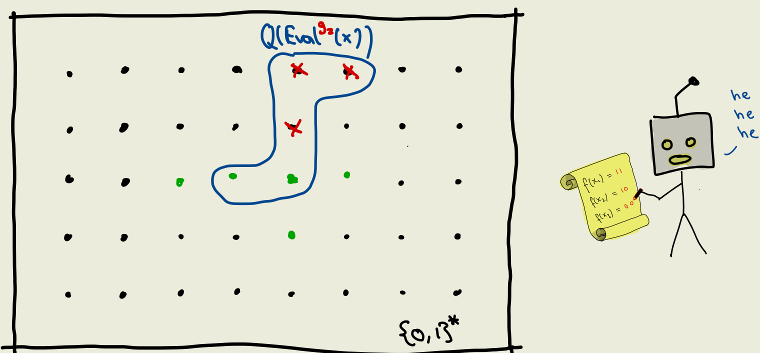


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.

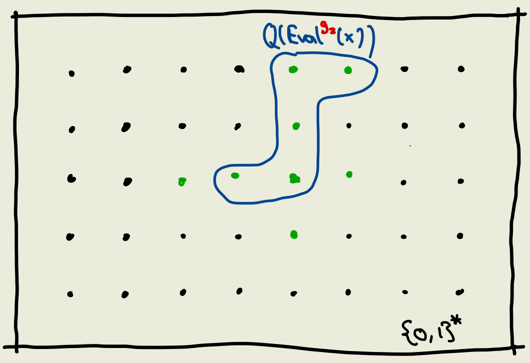


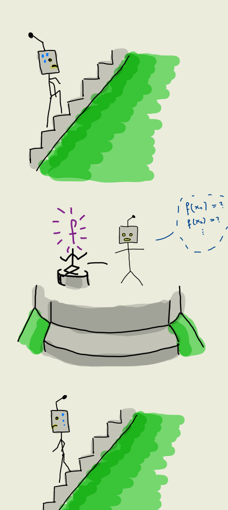
Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.



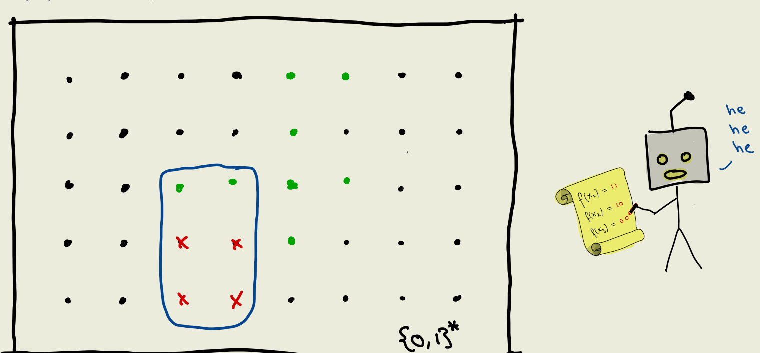


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.

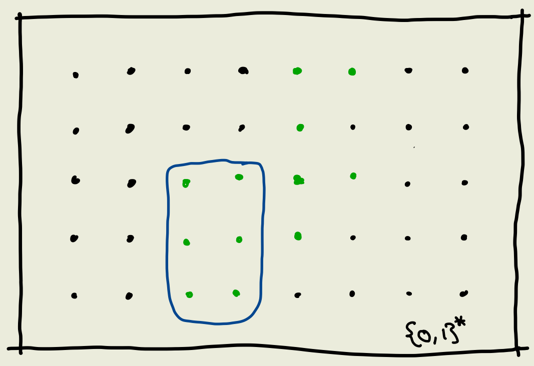


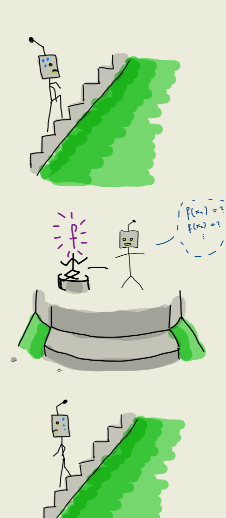
Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.



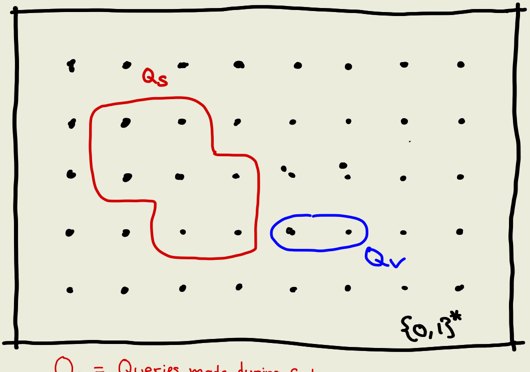


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.

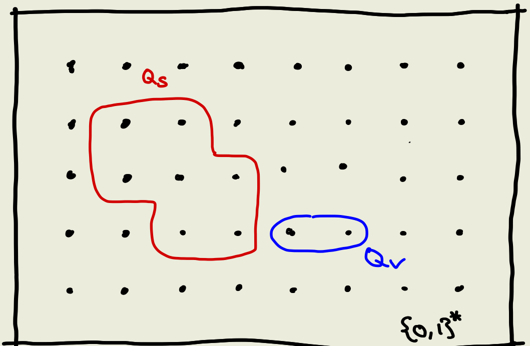


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Idea:

- Run Eval, and make up all of the queries.
- Query f on all queries afterwards.



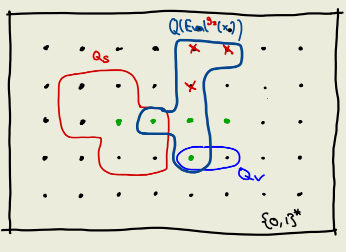
Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\mathrm{Setup}) \cup Q(\mathrm{Verify})$.

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in \color{red}{Q(\text{Setup})} \cup \color{blue}{Q(\text{Verify})}$.

two cases:



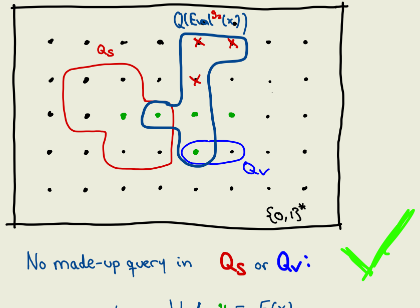
No made-up query in Q_S or Q_V:

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

two cases:



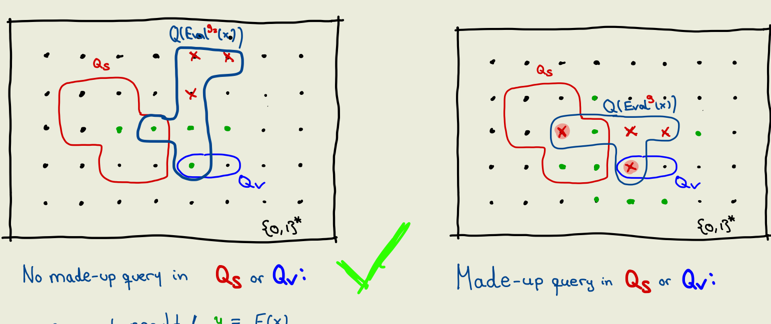
- correct result! $y = F(x)$

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

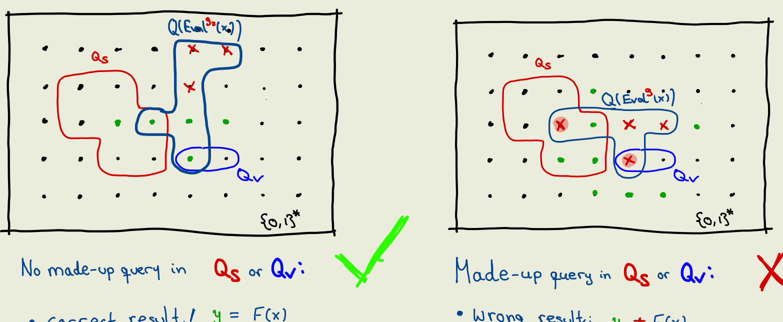
two cases:



Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

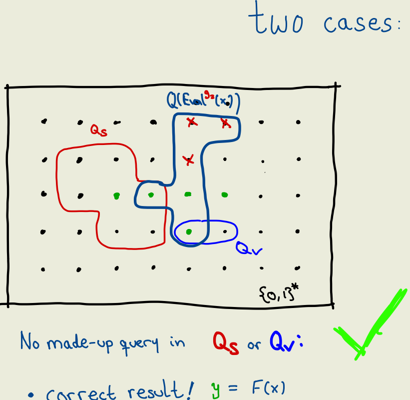


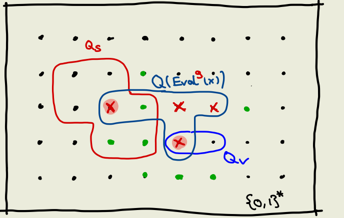
Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

two cases:





Made-up query in Qs or Qv:

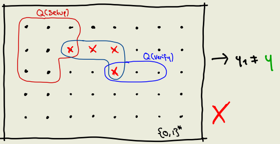
• Wrong result: y ≠ F(x)

• learn ≥1 new important query

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

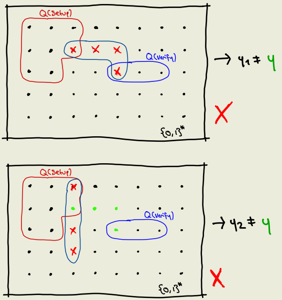


→ $y_1 \neq y$

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

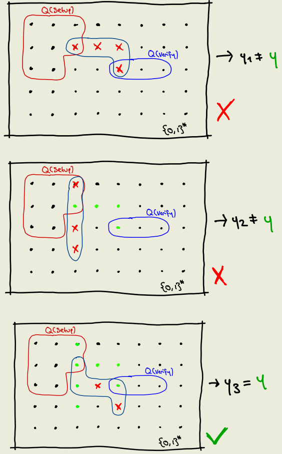
Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.



Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

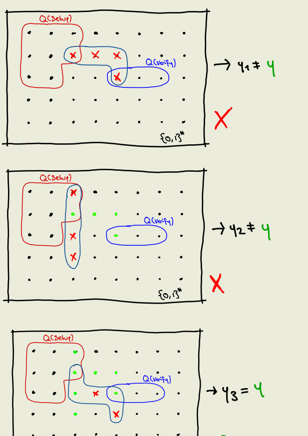
Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

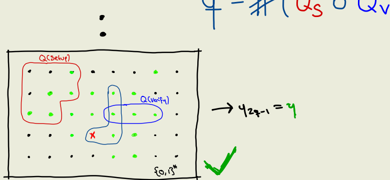


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.

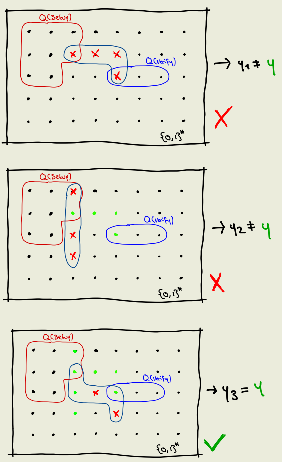


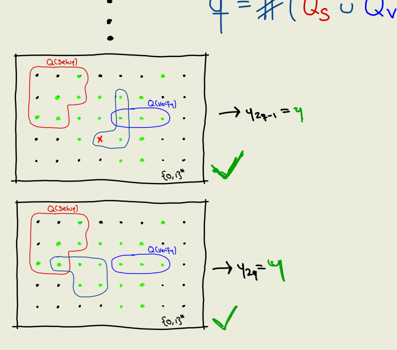


Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\mathrm{Setup}) \cup Q(\mathrm{Verify})$.

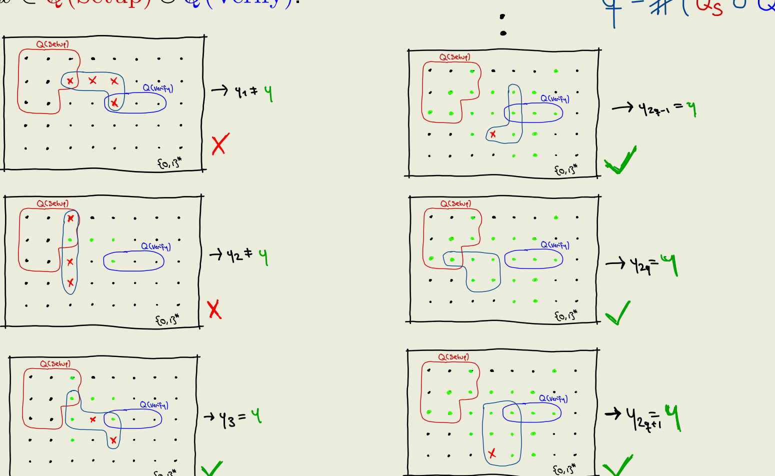




Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\text{Setup}) \cup Q(\text{Verify})$.



Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\mathrm{Setup}) \cup Q(\mathrm{Verify})$.

Theorem [MSW20]. If Π_{VDF} is *perfectly sound*, we can evaluate F in $O(q)$ rounds, where $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$.

Attack 2: the Dishonest Group; [MSW20]

Attacking perfectly sound VDFs

Observation: In each round, either win, or learn the value of f at a new input $x \in Q(\mathrm{Setup}) \cup Q(\mathrm{Verify})$.

Theorem [MSW20]. If Π_{VDF} is *perfectly sound*, we can evaluate F in $O(q)$ rounds, where $q = \#(Q(\mathrm{Setup}) \cup Q(\mathrm{Verify}))$.

Hence: If Setup makes $O(\log(T))$ queries, we break sequentiality in $O(\log(T))$ rounds.

Attack 3: the Extended Dishonest Group; [GRY25]

Attacking VDFs with deterministic Setup

Approach: construction of an algorithm $\mathsf{A}_{\mathrm{GRY}}$ that breaks soundness if $\mathsf{A}_{\mathrm{MSW}}$ fails to break sequentiality.

Attack 3: the Extended Dishonest Group; [GRY25]

Attacking VDFs with deterministic Setup

Approach: construction of an algorithm $\mathsf{A}_{\mathrm{GRY}}$ that breaks soundness if $\mathsf{A}_{\mathrm{MSW}}$ fails to break sequentiality.

Theorem [GRY25]. If Π_{VDF} has deterministic setup, we can evaluate F in $O(q)$ rounds, where $q = \#(Q(\mathrm{Verify}))$.

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$
[GRY25]	deterministic	usual	$O(\log(T))$

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$
[GRY25]	deterministic	usual	$O(\log(T))$

Open:

1. Can [MSW20] be extended to inefficient setup?

Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$
[GRY25]	deterministic	usual	$O(\log(T))$

Open:

1. Can [MSW20] be extended to inefficient setup?
2. Can [GRY25] be extended to probabilistic setup?

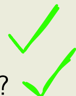
Overview of Previous Literature

$$q = \#Q(\text{Verify}) = O(\log(T))$$

Work	Setup	Soundness	# Rounds
[DGMV20]	no restrictions	usual	$T - O(T/q)$
[MSW20]	$\#Q \leq O(\log(T))$	perfect	$O(\log(T))$
[GRY25]	deterministic	usual	$O(\log(T))$
Our Work	no restrictions	usual	$O(\log(T))$

Open:

1. Can [MSW20] be extended to inefficient setup?
2. Can [GRY25] be extended to probabilistic setup?



Our work: Positive answer to both questions!

Extending [MSW20] to Expensive Setup

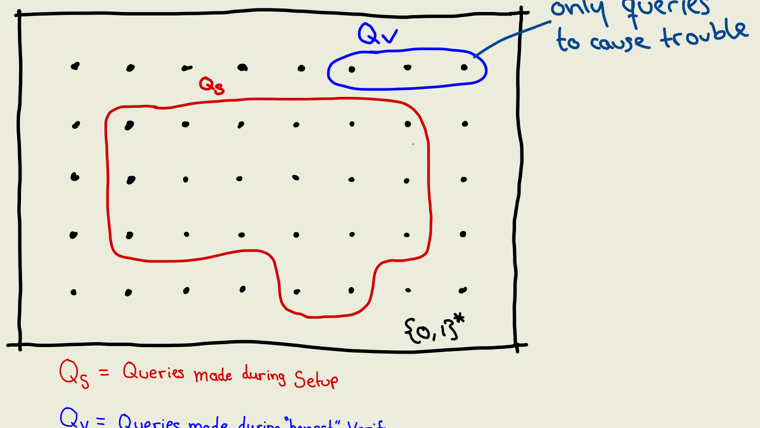
Main Observation:

If $\mathsf{A}^{f}_{\mathrm{MSW}}(\mathrm{pp}, x)$ does not make a (new) query at $x \in Q(\mathrm{Setup})$, then it computes $y = F(x)$ correctly in $O(\#Q(\mathrm{Verify}))$ rounds.

Extending [MSW20] to Expensive Setup

Main Observation:

If $\mathsf{A}^{f}_{\mathrm{MSW}}(\mathrm{pp}, x)$ *does not make a (new) query at* $x \in Q(\mathrm{Setup})$, *then it computes* $y = F(x)$ *correctly in* $O(\#Q(\mathrm{Verify}))$ *rounds.*



Extending [MSW20] to Expensive Setup

Main Observation:

If $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x)$ does not make a (new) query at $x \in Q(\mathrm{Setup})$, then it computes $y = F(x)$ correctly in $O(\#Q(\mathrm{Verify}))$ rounds.

after Preprocessing:

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   2.1 Sample $x \leftarrow X$ at random

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   2.1 Sample $x \leftarrow X$ at random
   2.2 Run $y \leftarrow \mathrm{Eval}^f(\mathrm{pp}, x)$

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   - 2.1 Sample $x \leftarrow X$ at random
   - 2.2 Run $y \leftarrow \mathrm{Eval}^f(\mathrm{pp}, x)$
   - 2.3 Store the results of all queries made in the previous step in I.

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   2.1 Sample $x \leftarrow X$ at random
   2.2 Run $y \leftarrow \mathrm{Eval}^f(\mathrm{pp}, x)$
   2.3 Store the results of all queries made in the previous step in I.
3. Return I.

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   - 2.1 Sample $x \leftarrow X$ at random
   - 2.2 Run $y \leftarrow \mathrm{Eval}^f(\mathrm{pp}, x)$
   - 2.3 Store the results of all queries made in the previous step in I.
3. Return I.

This ensures that $\mathrm{Eval}^f(\mathrm{pp}, x)$ (likely) does not make new queries to $Q(\mathrm{Setup})$.

Extending [MSW20] to Expensive Setup

Recall: construction of [DGMV20]:

simulate executions of $\mathrm{Eval}^f(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$: Let $q_s = \#Q(\mathrm{Setup})$

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   2.1 Sample $x \leftarrow X$ at random
   2.2 Run $y \leftarrow \mathrm{Eval}^f(\mathrm{pp}, x)$
   2.3 Store the results of all queries made in the previous step in I.
3. Return I.

This ensures that $\mathrm{Eval}^f(\mathrm{pp}, x)$ (likely) does not make new queries to $Q(\mathrm{Setup})$.

We want $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x, I)$ not to make queries to $Q(\mathrm{Setup})$.

Extending [MSW20] to Expensive Setup

Our Construction:

simulate executions of $\mathsf{A}^{f}_{\mathrm{MSW}}(\mathrm{pp}, x)$ for $x \leftarrow X$

Extending [MSW20] to Expensive Setup

Our Construction:

simulate executions of $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$:

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   - 2.1 Sample $x \leftarrow X$ at random
   - 2.2 Run $y \leftarrow \mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x, I)$
   - 2.3 Store the results of all queries made in the previous step in I.
3. Return I.

This ensures that $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x, I)$ (likely) does not make new queries to $Q(\mathrm{Setup})$.

Extending [MSW20] to Expensive Setup

Our Construction:

simulate executions of $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x)$ for $x \leftarrow X$

$\mathsf{P}^f(\mathrm{pp})$:

1. Set $I = \emptyset$, sample $t \leftarrow [2q_s]$ at random
2. For $i = 1, \ldots, t$:
   - 2.1 Sample $x \leftarrow X$ at random
   - 2.2 Run $y \leftarrow \mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x, I)$
   - 2.3 Store the results of all queries made in the previous step in I.
3. Return I.

This ensures that $\mathsf{A}^f_{\mathrm{MSW}}(\mathrm{pp}, x, I)$ (likely) does not make new queries to $Q(\mathrm{Setup})$.

this is what we wanted!

Overview of Results

Previously Open:

1. Can [MSW20] be extended to inefficient setup?
2. Can [GRY25] be extended to probabilistic setup?

Overview of Results

Previously Open:

1. Can [MSW20] be extended to inefficient setup? ✓
2. Can [GRY25] be extended to probabilistic setup?

Theorem 1. If Π_{VDF} is perfectly sound, then $(\mathsf{P}, \mathsf{A}_{\mathrm{MSW}})$ breaks sequentiality.

Overview of Results

Previously Open:

1. Can [MSW20] be extended to inefficient setup? ✓
2. Can [GRY25] be extended to probabilistic setup?

Theorem 1. If Π_{VDF} is perfectly sound, then $(\mathsf{P}, \mathsf{A}_{\mathrm{MSW}})$ breaks sequentiality.

Even More: we prove that the tools developed by [GRY25] also extend!

Overview of Results

Previously Open:

1. Can [MSW20] be extended to inefficient setup? ✓
2. Can [GRY25] be extended to probabilistic setup? ✓

Theorem 1. If Π_{VDF} is perfectly sound, then $(\mathsf{P}, \mathsf{A}_{\mathrm{MSW}})$ breaks sequentiality.

Even More: we prove that the tools developed by [GRY25] also extend!

Main Theorem. If Π_{VDF} is a sound VDF in the ROM, then it does not satisfy sequentiality.

Thank you!

Questions?

References

Döttling, Nico et al. "Tight Verifiable Delay Functions". In: *SCN 20*, pp. 65–84.

Guan, Ziyi, Artur Riazanov, and Weiqiang Yuan. "Breaking Verifiable Delay Functions in the Random Oracle Model". In: *CRYPTO 2025, Part VII*, pp. 161–191.

Mahmoody, Mohammad, Caleb Smith, and David J. Wu. "Can Verifiable Delay Functions Be Based on Random Oracles?" In: *ICALP 2020*, 83:1–83:17.