On the Security of Proofs of Sequential Work in a Post-Quantum World

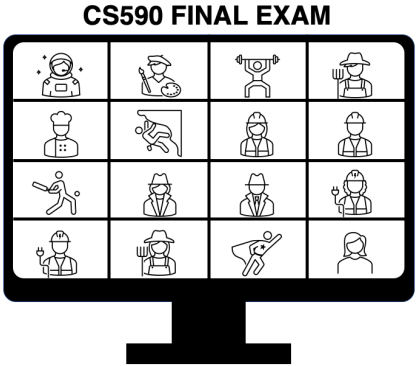
Jeremiah Blocki[1], Seunghoon Lee[1], Samson Zhou[2]

[1]Department of Computer Science, Purdue University
[2]School of Computer Science, Carnegie Mellon University

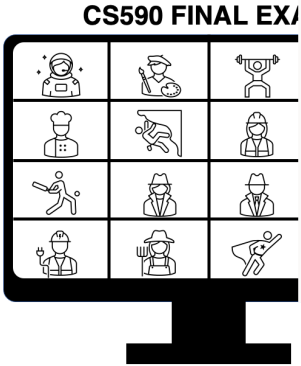
July 28, 2021

Conference on Information-Theoretic Cryptography (ITC) 2021

Motivation: Online Exams during the Pandemic



Motivation: Online Exams during the Pandemic



[CS590] 5 mins late - having internet issue

CG

Cinseer Goodman
Tue 5/2/2021 9:05 PM
To: Seunghoon Lee

answer-goodman.pdf
157 KB

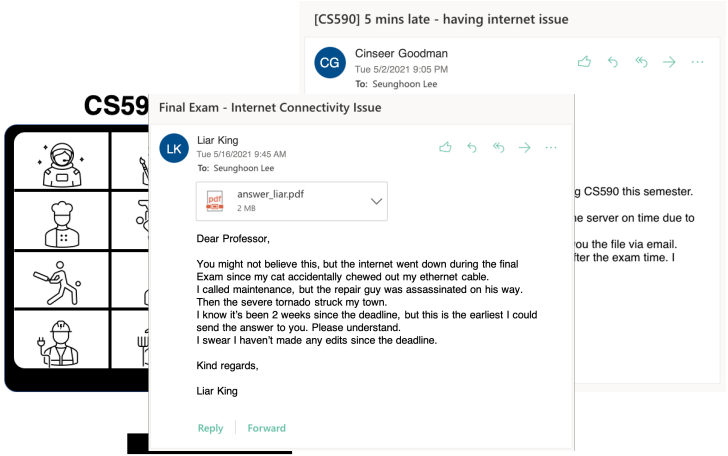
Dear Professor,

My name is Cinseer Goodman who is taking CS590 this semester.
I hope this email finds you well.
I was not able to submit the final exam to the server on time due to an unexpected internet connectivity loss.
It just went back 5 minutes later so I send you the file via email.
I promise I have not done any extra work after the exam time. I hope it works.
Thank you.

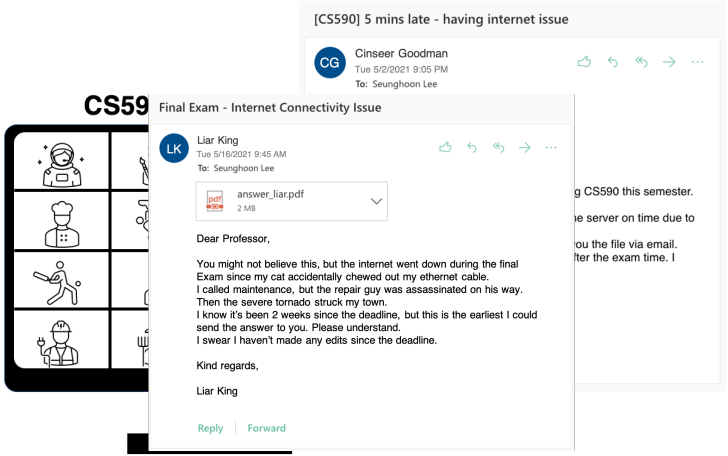
Best,
Cinseer Goodman

Reply | Forward

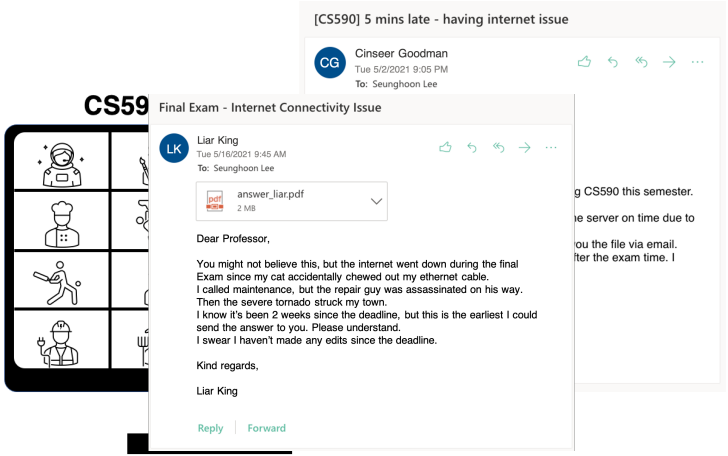
Motivation: Online Exams during the Pandemic



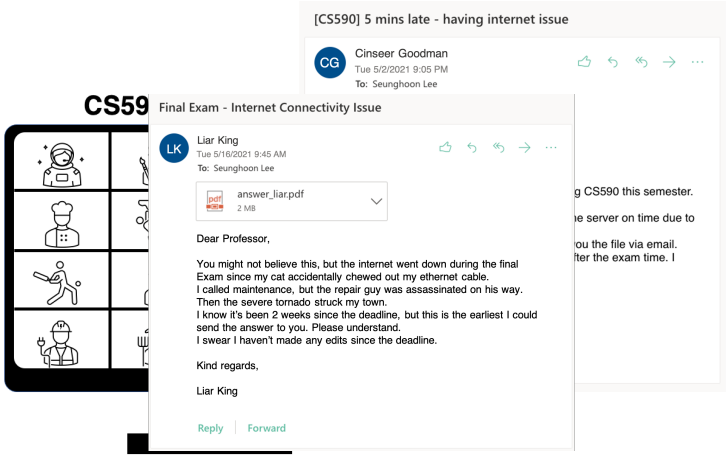
Motivation: Online Exams during the Pandemic



Motivation: Online Exams during the Pandemic



Motivation: Online Exams during the Pandemic



Solution: Proofs of Sequential Work (PoSW)

What is a Proof of Sequential Work? (Informal)

A proof that a large amount ( ) of sequential work was performed after a prover committed an initial message, e.g., the solution for the final exam

Solution: Proofs of Sequential Work (PoSW)

What is a Proof of Sequential Work? (Informal)

A proof that a large amount ( ) of sequential work was performed after a prover committed an initial message, e.g., the solution for the final exam

Initial approach: iterative hash chain

Solution: Proofs of Sequential Work (PoSW)

What is a Proof of Sequential Work? (Informal)

A proof that a large amount ( ) of sequential work was performed after a prover committed an initial message, e.g., the solution for the final exam

Initial approach: iterative hash chain

Disadvantage: Instructor needs to recompute the whole thing

Many late students? insufficient computational resources to verify all solutions

Solution: Proofs of Sequential Work (PoSW)

What is a Proof of Sequential Work? (Informal)

A proof that a large amount ( ) of sequential work was performed after a prover committed an initial message, e.g., the solution for the final exam

Initial approach: iterative hash chain

Disadvantage: Instructor needs to recompute the whole thing

Many late students? insufficient computational resources to verify all solutions

Additional requirements:

Efficiency: instructor (verifier) should be able to quickly verify the proofs (e.g., in time), and

Solution: Proofs of Sequential Work (PoSW)

What is a Proof of Sequential Work? (Informal)

A proof that a large amount ( ) of sequential work was performed after a prover committed an initial message, *e.g.*, the solution for the final exam

Initial approach: iterative hash chain

Disadvantage: Instructor needs to recompute the whole thing

Many late students? insufficient computational resources to verify all solutions

Additional requirements:

Efficiency: instructor (verifier) should be able to quickly verify the proofs (e.g., in time), and

Soundness: students (prover) should *not* be able to produce a *valid* proof faster (than sequential time , even if running in parallel).

PoSW Constructions

Mahmoody et al. [MMV13]: the first theoretical construction of a PoSW

- Verifier time , and prover time ,
- Parallel cheating prover running in sequential time cannot fool the verifier, and
- Security proof in the classical ROM.

PoSW Constructions

Mahmoody et al. [MMV13]: the first theoretical construction of a PoSW

- Verifier time , and prover time ,
- Parallel cheating prover running in sequential time cannot fool the verifier, and
- Security proof in the classical ROM.

Cohen and Pietrzak [CP18]: an improved & practical PoSW construction

- Modular security proof in the classical ROM:
  - Any parallel cheating prover (for the PoSW) must produce a long -sequence (whp), and
  - Any parallel cheating prover running in time $< N$ cannot produce an -sequence of length N (whp).

PoSW Constructions

Mahmoody et al. [MMV13]: the first theoretical construction of a PoSW

Verifier time , and prover time ,

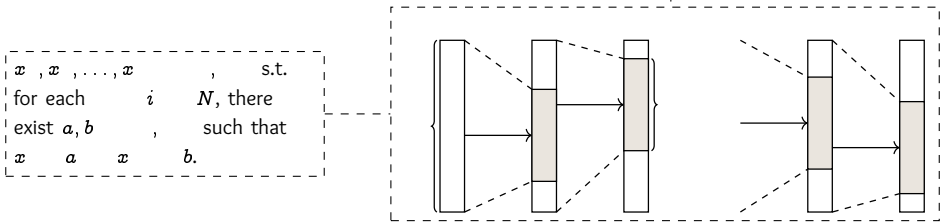
Parallel cheating prover running in sequential time cannot fool the verifier, and

Security proof in the classical ROM.

Cohen and Pietrzak [CP18]: an improved & practical PoSW construction

Modular security proof in the classical ROM:

Any parallel cheating prover (for the PoSW) must produce a long -sequence (whp), and
Any parallel cheating prover running in time $< N$ cannot produce an -sequence of length N (whp).



PoSW Constructions

Mahmoody et al. [MMV13]: the first theoretical construction of a PoSW

Verifier time , and prover time ,

Parallel cheating prover running in sequential time cannot fool the verifier, and

Security proof in the classical ROM.

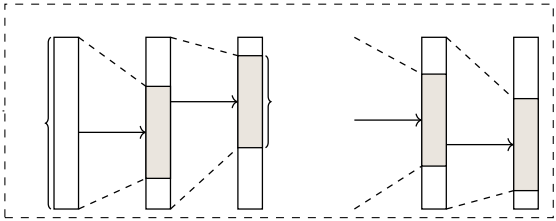
Cohen and Pietrzak [CP18]: an improved & practical PoSW construction

Post-Quantum Security?

Modular security proof in the classical ROM:

Any parallel cheating prover (for the PoSW) must produce a long -sequence (whp), and
Any parallel cheating prover running in time $< N$ cannot produce an -sequence of length N (whp).

x , x , . . . , x , s.t.
for each i N, there
exist a, b , such that
x a x b.



Post-Quantum Security of the PoSW

Cohen and Pietrzak [CP18]:

Any parallel cheating prover (for the PoSW) must produce a long -sequence,

Any parallel cheating prover running in time cannot produce an -sequence of length .

Post-Quantum Security of the PoSW

Cohen and Pietrzak [CP18]:

Any parallel cheating prover (for the PoSW) must produce a long -sequence,

Any parallel cheating prover running in time cannot produce an -sequence of length .

Key Research Questions:

Can a sequentially time-bounded parallel *quantum* attacker produce a long -sequence?

Can a sequentially time-bounded parallel *quantum* attacker produce a valid PoSW?

Post-Quantum Security of the PoSW

Cohen and Pietrzak [CP18]:

Any parallel cheating prover (for the PoSW) must produce a long -sequence,

Any parallel cheating prover running in time cannot produce an -sequence of length .

Key Research Questions:

Can a sequentially time-bounded parallel *quantum* attacker produce a long -sequence?

Can a sequentially time-bounded parallel *quantum* attacker produce a valid PoSW?

Short answer: NO!

Our Result. Hardness of Producing an -Sequence/PoSW in a Quantum Setting

Theorem (informal)

A quantum adversary making at most queries over rounds outputs an -sequence of length

(with where) with negligible probability —— .

Our Result. Hardness of Producing an -Sequence/PoSW in a Quantum Setting

Theorem (informal)

A quantum adversary making at most queries over rounds outputs an -sequence of length (with where) with negligible probability — .

Theorem (informal)

Suppose makes at most quantum queries to the random oracle over at most rounds. Then outputs a valid non-interactive PoSW with negligible probability — — .

Our Result. Hardness of Producing an -Sequence/PoSW in a Quantum Setting

Theorem (informal)

A quantum adversary making at most queries over rounds outputs an -sequence of length (with where) with negligible probability ——— .

Theorem (informal)

Suppose makes at most quantum queries to the random oracle over at most rounds. Then outputs a valid non-interactive PoSW with negligible probability ——— ——————— .

We give a direct proof for the non-interactive version of the PoSW from [CP18], and
Our proofs are in the parallel quantum random oracle model (pqROM).

Our Result. Hardness of Producing an -Sequence/PoSW in a Quantum Setting

Theorem (informal)

A quantum adversary making at most queries over rounds outputs an -sequence of length (with where) with negligible probability .

Theorem (informal)

Suppose makes at most quantum queries to the random oracle over at most rounds. Then outputs a valid non-interactive PoSW with negligible probability .

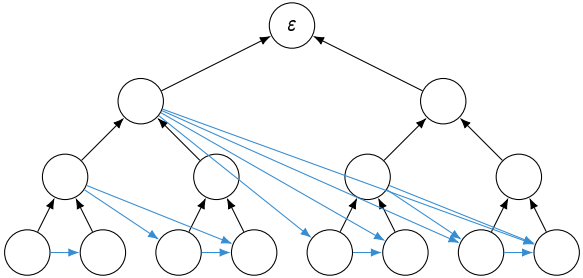
We give a direct proof for the non-interactive version of the PoSW from [CP18], and

Our proofs are in the parallel quantum random oracle model (pqROM).

Concurrent/Subsequent Work.

Chung et al. [CFHL21]: also gave a comparable security bounds for the PoSW in the pqROM

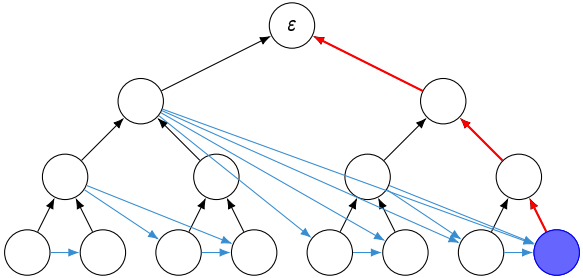
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

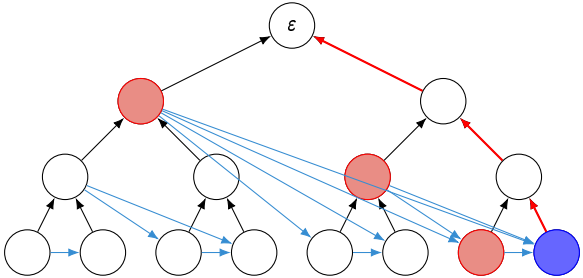
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

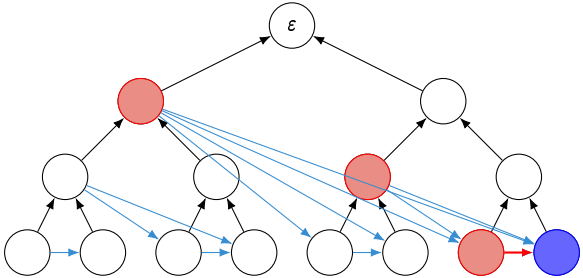
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

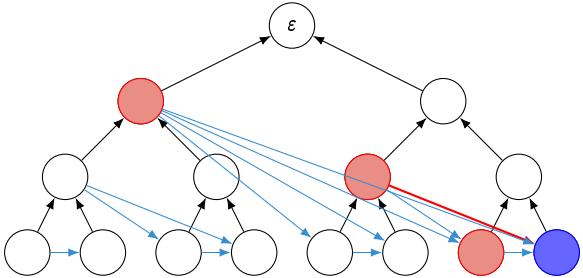
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

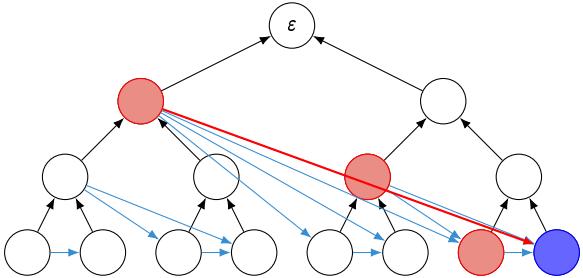
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

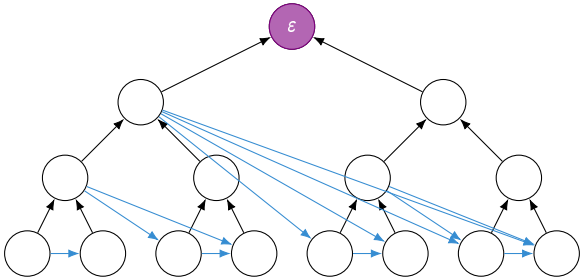
The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

The [CP18] Construction



statement being proved, e.g., final exam solution

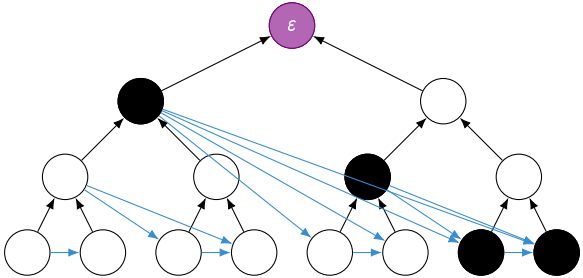
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved, e.g., final exam solution

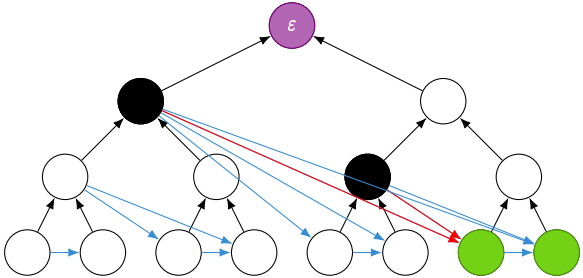
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved, e.g., final exam solution

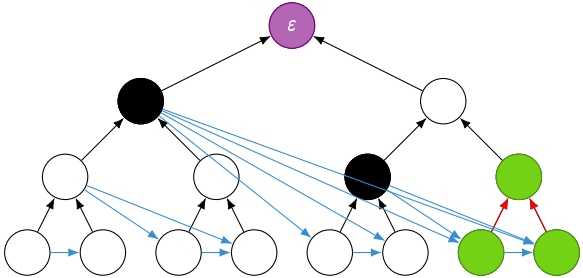
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved, e.g., final exam solution

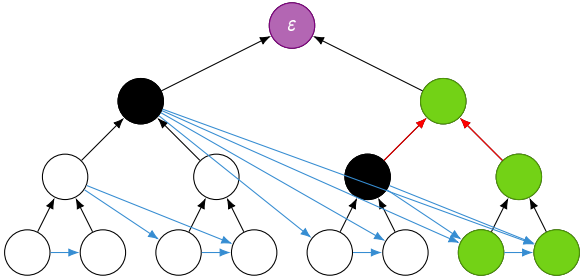
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved,
e.g., final exam solution

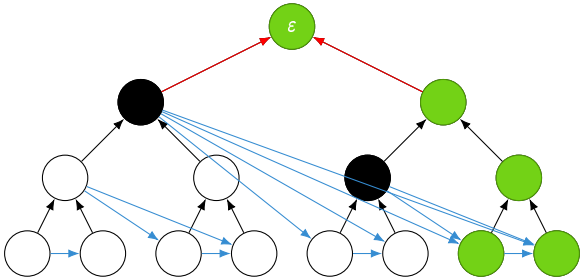
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved, e.g., final exam solution

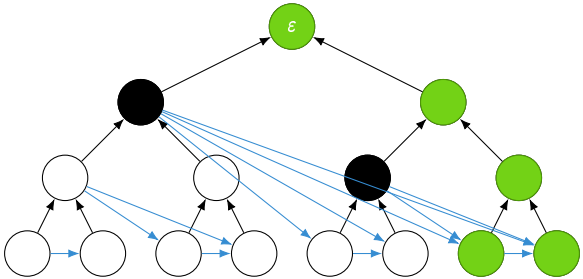
For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

The [CP18] Construction



statement being proved,
e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

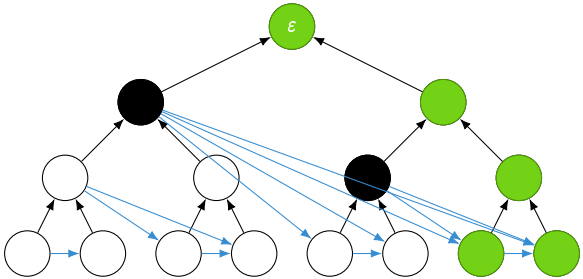
Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

Audit process: interactive or non-interactive (Fiat-Shamir)

The [CP18] Construction



statement being proved, e.g., final exam solution

For all leaf nodes , add an edge for any that is a left sibling of a node on the path from to the root

Each node has a label, a hash of its parents

The label of root node forms a Kcpijc rpcc amk kgrkclr of all the other nodes

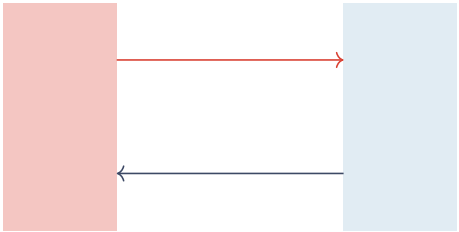
- Verifier can audit the prover by forcing the prover to open certain labels
- Show that they are locally consistent

Audit process: interactive or non-interactive (Fiat-Shamir)

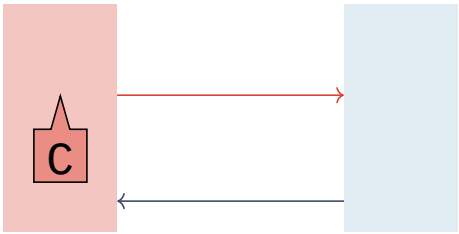
Any classical ROM attacker that produces a valid PoSW in time must produce a long -sequence

ROM vs qROM [BDF 11]

‹Classical ROM›

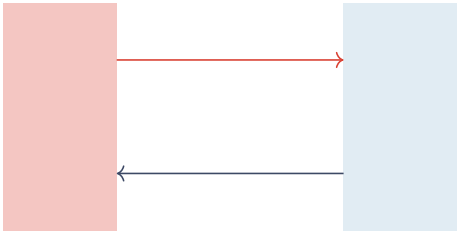


‹Quantum ROM›

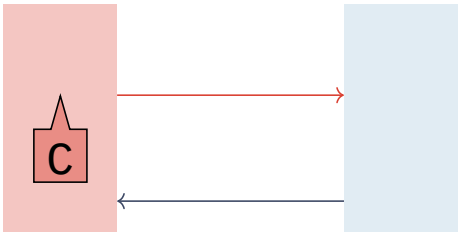


ROM vs qROM [BDF 11]

‹Classical ROM›



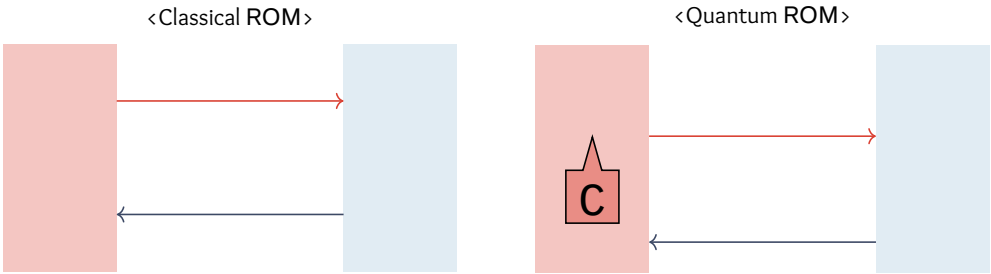
‹Quantum ROM›



Security proofs are much more challenging in the qROM

- Programmability & Extractability (ROM: 4, qROM: 8)
- Recording quantum queries?

ROM vs qROM [BDF 11]



Security proofs are much more challenging in the qROM

- Programmability & Extractability (ROM: 4, qROM: 8)
- Recording quantum queries?

Compressed Oracle Technique [Zha19]: change of view (compressed phase oracle (CPhsO))

Compressed Phase Oracle (CPhsO)

A database , where .

How to view a random oracle?

Classical: databases of known I/O pairs & unknown I/O pairs don't appear

Quantum: superposition over databases (known I/O pairs indeterminates)

Compressed Phase Oracle (CPhsO)

A database , where .

How to view a random oracle?

- Classical: databases of known I/O pairs & unknown I/O pairs don't appear
- Quantum: superposition over databases (known I/O pairs indeterminates)

After queries,

The state can be viewed as

where

Compressed Phase Oracle (CPhsO)

A database , where .

How to view a random oracle?

- Classical: databases of known I/O pairs & unknown I/O pairs don't appear
- Quantum: superposition over databases (known I/O pairs indeterminates)

After queries,

The state can be viewed as

where

query registers,

Compressed Phase Oracle (CPhsO)

A database , where .

How to view a random oracle?

- Classical: databases of known I/O pairs & unknown I/O pairs don't appear
- Quantum: superposition over databases (known I/O pairs indeterminates)

After queries,

The state can be viewed as

where

- query registers,
- auxiliary input (algorithm state), and

Compressed Phase Oracle (CPhsO)

A database , where .

How to view a random oracle?

- Classical: databases of known I/O pairs & unknown I/O pairs don't appear
- Quantum: superposition over databases (known I/O pairs indeterminates)

After queries,

The state can be viewed as

where

- query registers,
- auxiliary input (algorithm state), and
- a compressed dataset of at most input/output pairs.

Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

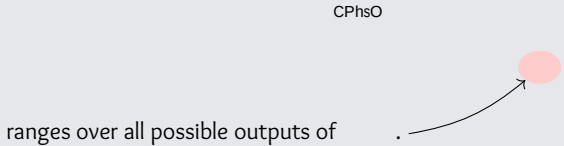
Example: Single Query (simplest case)

CPhsO

ranges over all possible outputs of .

Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

Example: Single Query (simplest case)

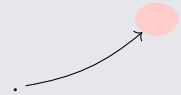


Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

Example: Single Query (simplest case)

CPhsO

ranges over all possible outputs of .



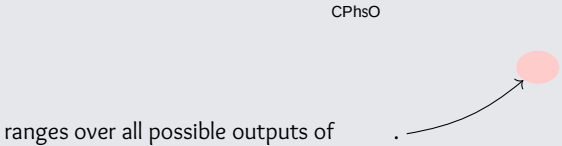
Example: Parallel Query (simplest case)

CPhsO

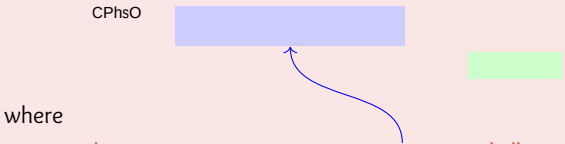
where

Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

Example: Single Query (simplest case)

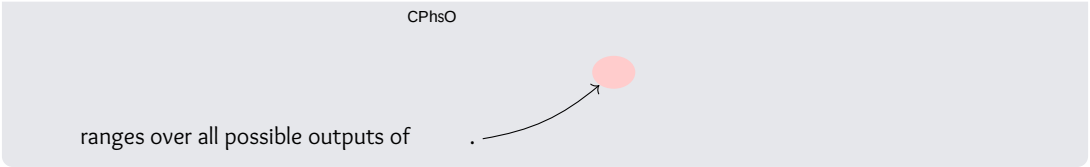


Example: Parallel Query (simplest case)

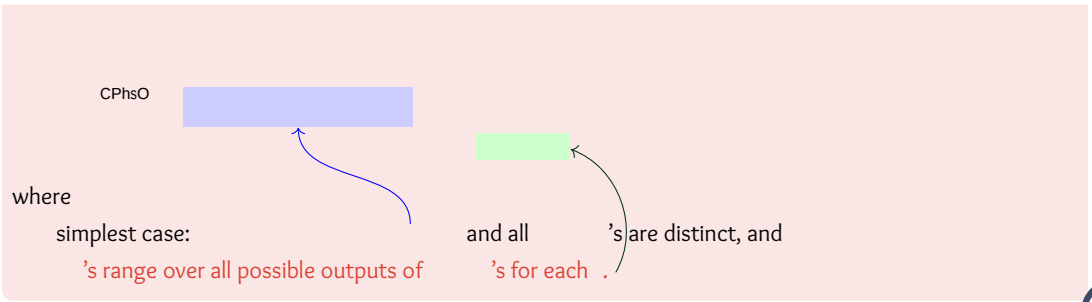


Extending Compressed Oracle Technique to the pqROM: the oracle CPhsO^k

Example: Single Query (simplest case)

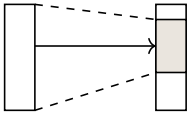


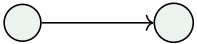
Example: Parallel Query (simplest case)



Notations

Given a database , define a directed graph on nodes such that:



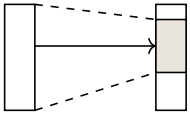


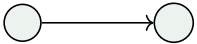
PATH contains a path of length (set of databases), and

PATH PATH (set of basis states).

Notations

Given a database , define a directed graph on nodes such that:





PATH contains a path of length (set of databases), and

PATH PATH (set of basis states).

contains an -sequence of length

PATH

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Interpretation/Intuition:

PATH : 2-norm of the projection of onto PATH , i.e.,

PATH

PATH

If we start with the state that is nearly orthogonal to PATH , then after applying the oracle CPhsO , the resulting state is also nearly orthogonal to PATH .

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Basic proof idea: split the states into good and bad part. (in this talk, suppose that and all distinct for simplicity)

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Basic proof idea: split the states into good and bad part. (in this talk, suppose that and all distinct for simplicity)

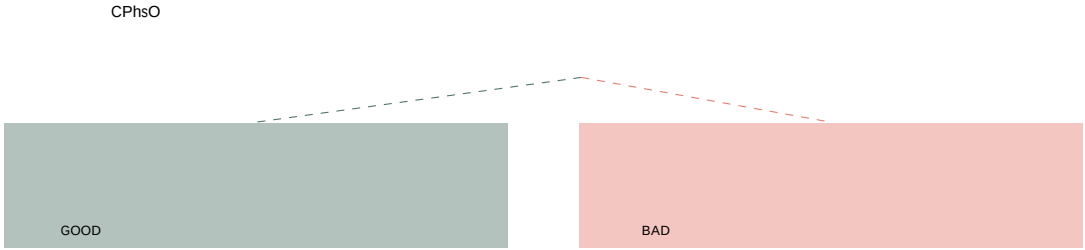
CPhsO

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Basic proof idea: split the states into good and bad part. (in this talk, suppose that and all distinct for simplicity)

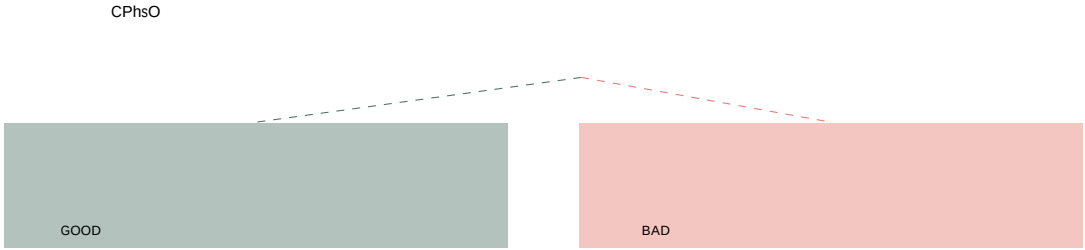


Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Basic proof idea: split the states into good and bad part. (in this talk, suppose that and all distinct for simplicity)



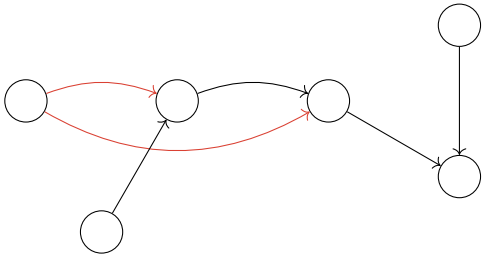
! : PATH but PATH

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Proof by example:

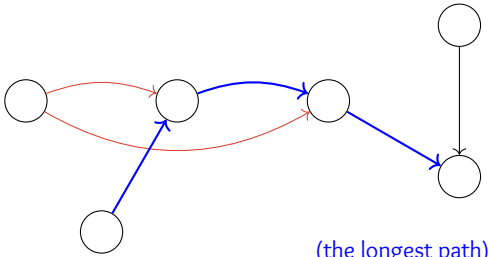


Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Lemma

: an initial state, and let CPhsO *. Then* PATH PATH .

Proof by example:

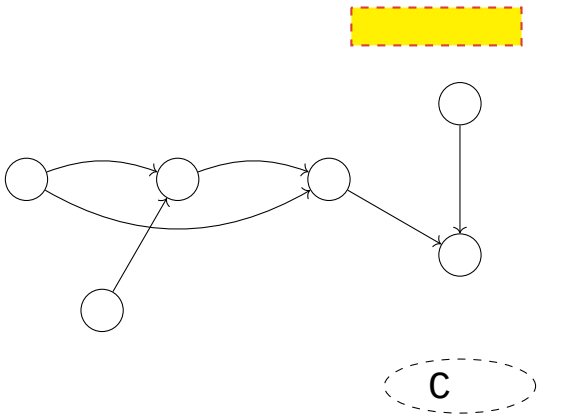


(the longest path)=

PATH but PATH

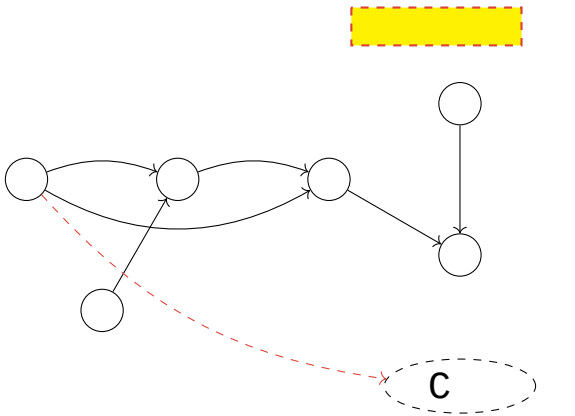
Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Suppose we have one query: (where). Then the updated database is:



Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

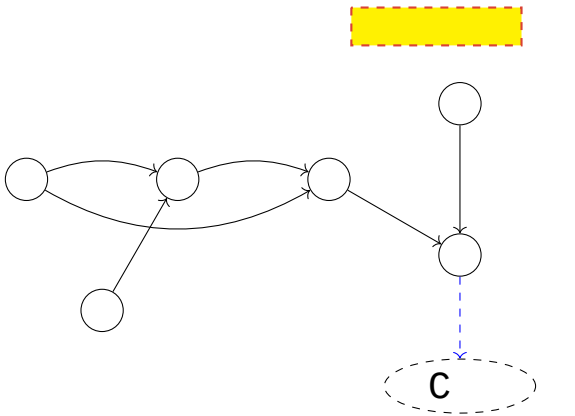
Suppose we have one query: (where). Then the updated database is:



superposition over 's

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

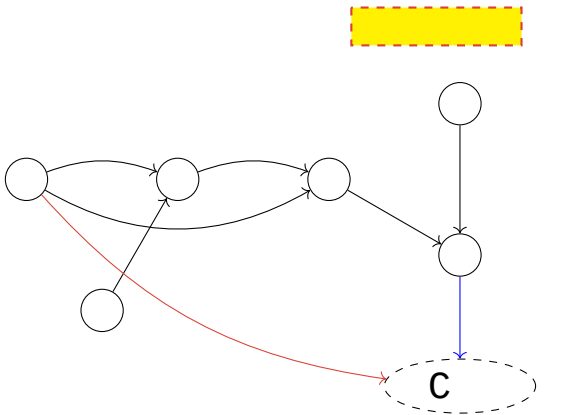
Suppose we have one query: (where). Then the updated database is:



superposition over 's

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

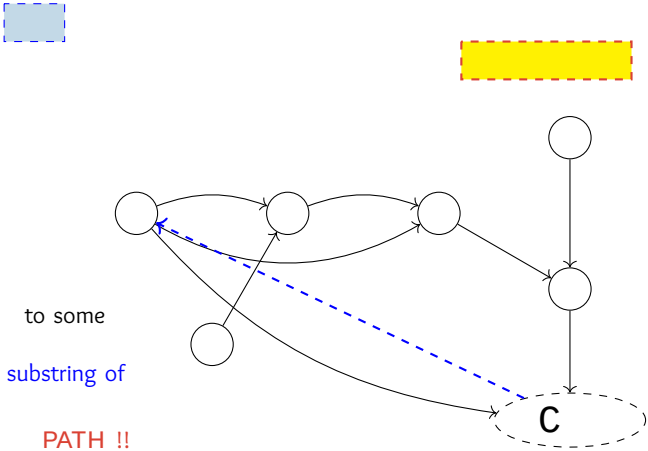
Suppose we have one query: (where). Then the updated database is:



superposition over 's

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Suppose we have one query: (where). Then the updated database is:



! if:

1. back edges from to some .

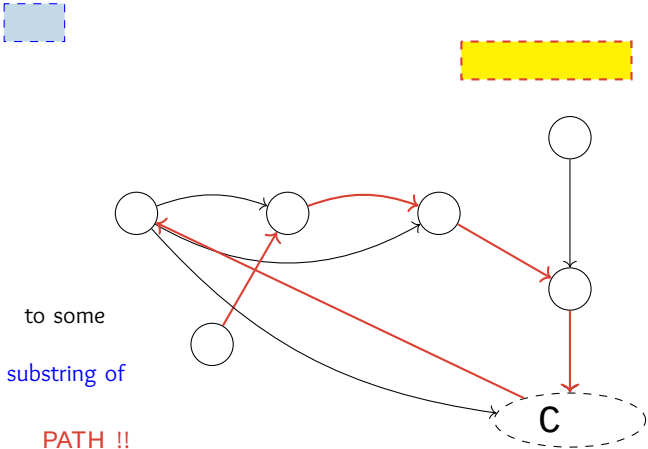
(e.g., substring of)

PATH but PATH !!

superposition over 's

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Suppose we have one query: (where). Then the updated database is:



! if:

1. back edges from to some .

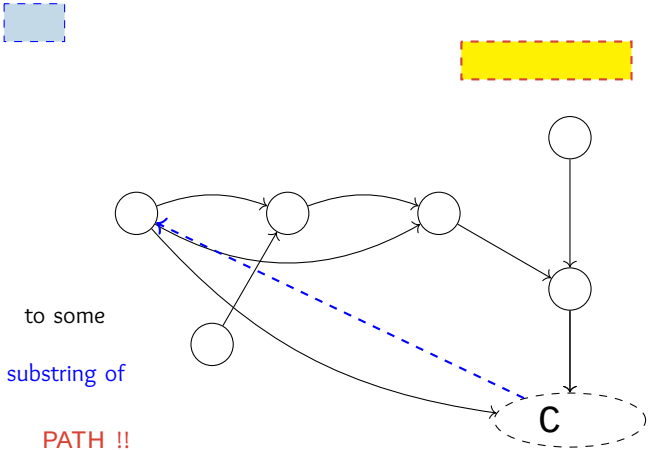
(e.g., substring of)

PATH but PATH !!

superposition over 's

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

Suppose we have one query: (where). Then the updated database is:



! if:

1. back edges from to some .

(e.g., substring of)

PATH but PATH !!

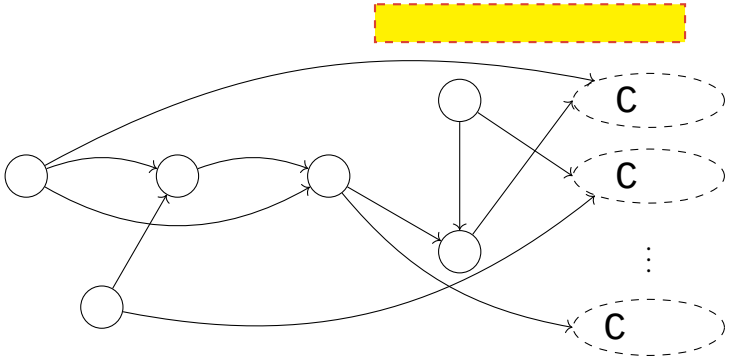
Key observation:

The fraction of such 's is negligible! (out of)

superposition over 's

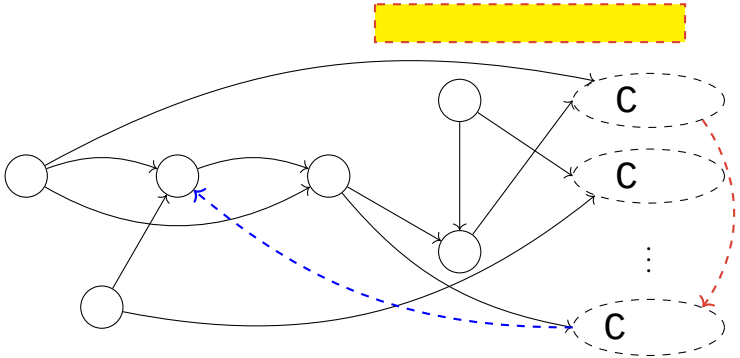
Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

For a parallel query: (where and all 's are distinct for simplicity),



Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

For a parallel query: (where and all 's are distinct for simplicity),



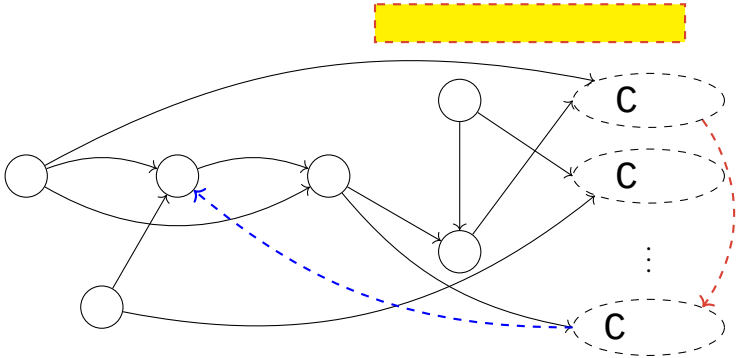
! if:

1. internal edges between 's, and
2. back edges from to some .

PATH but PATH !!

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

For a parallel query: (where and all 's are distinct for simplicity),



! if:

1. internal edges between 's, and
2. back edges from to some .

PATH but PATH !!

Key observation: The fraction of such 's is negligibly small! (out of for each)

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

We have shown: parallel queries in a single round,

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

We have shown: parallel queries in a single round,

PATH PATH

Throughout rounds: parallel queries in each round, with .

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

We have shown: parallel queries in a single round,

PATH PATH

Throughout rounds: parallel queries in each round, with .

By triangle inequality,

PATH

measures a database in PATH with probability at most ,

i.e., can produce an -sequence of length with only negligible probability .

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

We have shown: parallel queries in a single round,

PATH PATH

Throughout rounds: parallel queries in each round, with .

By triangle inequality,

PATH

measures a database in PATH with probability at most ,

i.e., can produce an -sequence of length with only negligible probability .

We only considered the simplest case in this talk - see the paper for the full security proof!

Proof Ideas: Hardness of Producing an -sequence in a Quantum Setting

We have shown: parallel queries in a single round,

PATH PATH

Throughout rounds: parallel queries in each round, with .

By triangle inequality,

PATH

measures a database in PATH with probability at most ,

i.e., can produce an -sequence of length with only negligible probability .

We only considered the simplest case in this talk - see the paper for the full security proof!

Security of a non-interactive PoSW: similar argument using the result above - details in the paper

(M f f K f f < 8 8 @ K ; 8 C A ⊁ K

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability —— .

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Open Questions.

Can we tighten the security bound from to ?

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Open Questions.

Can we tighten the security bound from to ?

Establishing security for larger : can we extract more than challenges from a single RO output?

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Open Questions.

Can we tighten the security bound from to ?

Establishing security for larger : can we extract more than challenges from a single RO output?

Can we prove for an interactive PoSW?

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Open Questions.

Can we tighten the security bound from to ?

Establishing security for larger : can we extract more than challenges from a single RO output?

Can we prove for an interactive PoSW?

Can we extend our security proof for other PoSW constructions?

Concluding Remarks

Takeaways.

PoSW allows a prover to convince a resource-bounded verifier that the prover invested a substantial amount of sequential time to produce a valid proof.

Any attacker in the pqROM making total queries in sequential rounds cannot find an -sequence of length except with negligible probability .

Any attacker in the pqROM making total queries in sequential time cannot produce a valid non-interactive PoSW ([CP18] construction) except with negligible probability .

Open Questions.

Can we tighten the security bound from to ?

Establishing security for larger : can we extract more than challenges from a single RO output?

Can we prove for an interactive PoSW?

Can we extend our security proof for other PoSW constructions?

Can techniques extend to other primitives, e.g., Proofs of Space, Memory-Hard Functions, etc.?

References I

- Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry, *Random oracles in a quantum world*, ASIACRYPT 2011 (Dong Hoon Lee and Xiaoyun Wang, eds.), LNCS, vol. 7073, Springer, Heidelberg, December 2011, pp. 41–69.
- Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao, *On the compressed-oracle technique, and post-quantum security of proofs of sequential work*, 2021.
- Bram Cohen and Krzysztof Pietrzak, *Simple proofs of sequential work*, EUROCRYPT 2018, Part II (Jesper Buus Nielsen and Vincent Rijmen, eds.), LNCS, vol. 10821, Springer, Heidelberg, April / May 2018, pp. 451–467.
- Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan, *Publicly verifiable proofs of sequential work*, ITCS 2013 (Robert D. Kleinberg, ed.), ACM, January 2013, pp. 373–388.
- Mark Zhandry, *How to record quantum queries, and applications to quantum indifferentiability*, CRYPTO 2019, Part II (Alexandra Boldyreva and Daniele Micciancio, eds.), LNCS, vol. 11693, Springer, Heidelberg, August 2019, pp. 239–268.