Adaptively Secure Succinct Garbled RAM with Persistent Memory

Ran Canetti, Yilei Chen, Justin Holmgren, Mariana Raykova

DIMACS workshop
MIT Media Lab
June 8~10, 2016
June 11, 2016, Boston, heavy snow.
June 11, 2016, Boston, heavy snow. Alice finds a quasi-polynomial time algorithm for factoring.
June 11, 2016, Boston, heavy snow. Alice finds a quasi-polynomial time algorithm for factoring.
June 11, 2016, Boston, heavy snow. Alice finds a quasi-polynomial time algorithm for factoring. Instead of submitting to STOC, she thinks it’s cool to write a program and show off to her friends.
> Factoring.hs RSA2048
> Factoring.hs  RSA2048
Running time 7 hrs 34 mins
25195908475…20720357
= 83990...4079279 x 3091701...723883

Next question
: It is slow on her laptop (quasi-polynomial time, you know) … cannot fit into a party.
It is slow on her laptop (quasi-polynomial time, you know) ... cannot fit into a party.

So she turns to cloud, but clouds are big brothers
It is slow on her laptop (quasi-polynomial time, you know) … cannot fit into a party.

So she turns to cloud, but clouds are big brothers.

She heard that one can delegate the computation in a way that the server learns only the output of the computation but nothing else.
“My friends and NSA will be shocked by the runtime without learning anything other than the output”
“The algorithm has huge preprocessing, stores lots of non-zero points on the Zeta function ...”

“My friends and NSA will be shocked by the runtime without learning anything other than the output”
“The algorithm has huge preprocessing, stores lots of non-zero points on the Zeta function ...”

“My friends and NSA will be shocked by the runtime without learning anything other than the output”

“Wait ... the audiences already know too much.”
> sudo apt-get install FHE
> sudo apt-get install  FHE
> FHE  Factoring.hs
> sudo apt-get install FHE
> FHE Factoring.hs

Turning the program into circuits ...
> sudo apt-get install FHE
> FHE Factoring.hs

Turning the program into circuits ...

^C
> sudo apt-get install   FHE
> FHE   Factoring.hs

Turning the program into circuits ...

^C

>

> sudo apt-get install  Yao
> Yao   Factoring.hs
> sudo apt-get install FHE
> FHE Factoring.hs
Turning the program into circuits ...
^C
>
> sudo apt-get install Yao
> Yao Factoring.hs
Still turning the program into circuits ...

#Yao
> sudo apt-get install FHE
> FHE Factoring.hs
Turning the program into circuits ...
^C
>
> sudo apt-get install Yao
> Yao Factoring.hs
Still turning the program into circuits ...
^C^C^C^C^C^C^C^C
>
> sudo apt-get install GRAM_Lu_Ostrovsky
> GRAM_Lu_Ostrovsky Factoring.hs
> sudo apt-get install GRAM_Lu_Ostrovsky
> GRAM_Lu_Ostrovsky Factoring.hs

Warning: Program size as big as the running time, continue (y) or not (n)
> sudo apt-get install GRAM_Lu_Ostrovsky
> GRAM_Lu_Ostrovsky Factoring.hs
Warning: Program size as big as the running time, continue (y) or not (n)
n
>
> sudo apt-get install PRAM
> sudo apt-get install PRAM
> PRAM Factoring.hs
> sudo apt-get install PRAM
> PRAM Factoring.hs
Done -> PRAM_Factoring
> sudo apt-get install  PRAM
> PRAM  Factoring.hs
Done -> PRAM_Factoring
> PRAM_Factoring  RSA2048
> sudo apt-get install PRAM
> PRAM Factoring.hs
Done -> PRAM_Factoring
> PRAM_Factoring RSA2048
Warning: cannot adaptively choose functions or inputs, security at user’s own risk, continue (y) or not (n)
> sudo apt-get install PRAM
> PRAM Factoring.hs
Done -> PRAM_Factoring
> PRAM_Factoring RSA2048
Warning: cannot adaptively choose functions or inputs, security at user’s own risk, continue (y) or not (n)
n
“Huge amount of preprocessed data reusable”

“Don’t turn into circuits, don’t blow up too much”

“Adaptively pick integers”
Garbling/randomized encoding for RAM with persistent memory
Garbling/randomized encoding for RAM with persistent memory

Gen => msk
Garbling/randomized encoding for RAM with persistent memory

\[ \text{Gen} \rightarrow \text{msk} \]

\[ \text{msk} + D_0 \rightarrow G(D_0) \]
Garbling/randomized encoding for RAM with persistent memory

\[
\text{Gen} \Rightarrow \text{msk}
\]

\[
\text{msk} + D_0 \Rightarrow G(D_0)
\]

\[
\text{msk} + P_1 \Rightarrow G(P_1)
\]
Garbling/randomized encoding for RAM with persistent memory

\[ \text{Gen} \Rightarrow \text{msk} \]

\[
\text{msk} + D_0 \Rightarrow G(D_0)
\]

\[
\text{msk} + P_1 \Rightarrow G(P_1)
\]

\[ \text{Eval} \quad G(D_0) \quad G(P_1) \quad \Rightarrow \quad P_1(D_0) \]
Garbling/randomized encoding for RAM with persistent memory

Gen => msk

msk + D₀ => G(D₀)

msk + P₁ => G(P₁)

Eval
G(D₀)  G(P₁) => P₁(D₀)  G(D₁)

Persistency
Garbling/randomized encoding for RAM with persistent memory

**Gen** => **msk**

**msk** + **D₀** => **G(D₀)**

**msk** + **P₁** => **G(P₁)**

**Eval**

**G(D₀)** + **G(P₁)** => **P₁(D₀)**

**msk** + **P₂** => **G(P₂)**

**Persistency**
Garbling/randomized encoding for RAM with persistent memory

\[
\text{Gen} \Rightarrow \text{msk}
\]

\[
\text{msk} + D_0 \Rightarrow G(D_0)
\]

\[
\text{msk} + P_1 \Rightarrow G(P_1)
\]

\[
\text{Eval} \quad G(D_0) \quad G(P_1) \Rightarrow P_1(D_0) \quad G(D_1)
\]

\[
\text{Eval} \quad G(D_1) \quad G(P_2) \Rightarrow P_2(D_1) \quad G(D_2)
\]

...
Garbling/randomized encoding for RAM with persistent memory

\[ G(D_0) \]

\[ G(P_1) \]

Succinct
Garbling/randomized encoding for RAM with persistent memory

\[ P_1(D_0) \]

Adaptively simulation secure
Garbling/randomized encoding for RAM with persistent memory

\[ ? \rightarrow G(D_0) \]
\[ ? \rightarrow G(P_1) \]
\[ G(D_0) \quad G(P_1) \leq P_1(D_0) \quad G(D_1) \]

Adaptively simulation secure
Garbling/randomized encoding for RAM with persistent memory

\[ ? \implies G(D_0) \]
\[ ? \implies G(P_1) \]
\[ G(D_0) \leq G(D_1) \]

Adaptively simulation secure
Garbling/randomized encoding for RAM with persistent memory

\[ \begin{align*}
? & \Rightarrow G(D_0) \\
? & \Rightarrow G(P_1) \\
G(D_0) & \Rightarrow G(P_1) \leq P_1(D_0) \Rightarrow G(D_1) \\
? & \Rightarrow G(P_2) \\
G(D_1) & \Rightarrow G(P_2) \leq P_2(D_1) \Rightarrow G(D_2)
\end{align*} \]

Adaptively simulation secure
Theorem
[Main Theorem]

Adaptively secure succinct garbled RAM with persistent memory from indistinguishability obfuscation for circuits, and poly-to-1 collision-resistant hash function.
Starring
Indistinguishability Obfuscator
Indistinguishability Obfuscator for circuits

Defined by [Barak-Goldreich-Impagliazzo-Rudich-Sahai-Vadhan-Yang ’01]

Security:

\[ iO[ F_0 ] \approx iO[ F_1 ] \]

if \( F_0 \) and \( F_1 \) have identical functionality

Candidate constructions:
[Garg-Gentry-Halevi-Raykova-Sahai-Waters ‘13], [Barak-Garg-Kalai-Paneth-Sahai ‘14],
[Brakerski-Rothblum ‘14], [Pass-Seth-Telang ‘14], [Zimmerman ‘15], [Applebaum-Brakerski ‘15],
[Ananth-Jain ‘15], [Bitansky-Vaikuntanathan ‘15], [Gentry-Gorbunov-Halevi ‘15], [Lin ‘16], …

Cryptanalyses:
[Cheon-Han-Lee-Ryu-Stehle ‘15], [Coron et al ‘15], [Miles-Sahai-Zhandry ‘16], …
Poly-to-one
Collision Resistant Hash function
Poly-to-one collision resistant hash functions

H is collision resistant + each image has at most poly preimages.

[Thm] Exists for constant c, assuming Factoring or Discrete-log is hard.
The rest of the talk:

1. The main idea of the construction.
2. The technical heart: adaptively-enforceable accumulator.
3. Wrap up, and the easiest ways to think of our scheme.
Starting point: Canetti-Holmgren’s selectable secure scheme.
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.
You never know how hard it is to use iO before actually play with it.

[ said Justin Holmgren, June 22, 2015, sunny ]
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
**Fixed-transcript** => Fixed-access => Fixed-address => Fully secure

Indistinguishable as long as \(\text{transc} = (q, \text{op})\) are the same.

[KLW-technique]
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
Fixed-transcript $\Rightarrow$ Fixed-access $\Rightarrow$ Fixed-address $\Rightarrow$ Fully secure

- Indistinguishable as long as $\text{transc} = (q, \text{op})$ are the same.
- $q$ can be different
- [KLW-technique] [encrypt the state]
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

Indistinguishable as long as \(\text{transc} = (q, \text{op})\) are the same. [KLW-technique]
q can be different [encrypt the state]
Memory content can be different [encrypt the data]
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

Indistinguishable as long as $\text{transc} = (q, \text{op})$ are the same. [KŁW-technique]
q can be different [encrypt the state]
Memory content can be different [encrypt the data]
Hide access pattern. [oram]
Starting point: Canetti-Holmgren’s selective secure scheme.

Garble the CPU-step circuit, encrypt and authenticate the intermediate states, memories.

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

Indistinguishable as long as $\text{transc} = (q, \text{op})$ are the same. [KLW-technique]

$q$ can be different [encrypt the state]

Memory content can be different [encrypt the data]

Hide access pattern. [oram]
Canetti-Holmgren (ITCS16)
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
Canetti-Holmgren (ITCS16)  
+ Zoom-in the core step:  

Koppula-Lewko-Waters (STOC15)  
(iO-friendly) Iterator  
(iO-friendly) Accumulator  
(iO-friendly) Splittable signature
Canetti-Holmgren (ITCS16) + Zoom-in the core step:

Koppula-Lewko-Waters (STOC15) (iO-friendly) Iterator
(iO-friendly) Accumulator (iO-friendly) Splittable signature

Accumulator iO-friendly Merkle-tree

What is written in eprint 2015/1074

Algorithm 11: Transition function for $M_i$ with memory verified by a signed accumulator.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:

Koppula-Lewko-Waters (STOC15)
(iO-friendly) Iterator
(iO-friendly) Accumulator
(iO-friendly) Splittable signature

What is written in eprint 2015/1074
Canetti-Holmgren (ITCS16) + Zoom-in the core step:


What is written in eprint 2015/1074

Algorithm 1: Transition function for $M_i$, with memory verified by a signed accumulator.
Canetti-Holmgren (ITCS16) + Zoom-in the core step:


Accumulator
iO-friendly Merkle-tree

- Initialize
- Authenticate
- Update

G(D_0)

G(P_{i+1})

G(D_{i+1})

key

Note: The diagram represents a simplified view of the accumulator and its operations. The specific details and formulas mentioned are not shown in the diagram. The text refers to "iO-friendly" and "splittable" signatures, which are important in the context of zero-knowledge proofs and secure computation.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.

#Merkletree
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.

#Merkletree
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
- **Enforcement (iO-friendly property):** there’s only one preimage $x^*$ of the current root value $y^*$.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
- **Enforcement (iO-friendly property):** there’s only one preimage $x^*$ of the current root value $y^*$.

Impossible information theoretically.
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
- Enforcement (iO-friendly property): there’s only one preimage \( x^* \) of the current root value \( y^* \).

Impossible information theoretically.

KLW’s computational enforcement:  
Normal.Gen(            )->H  
Enforce.Gen( \( x^*, y^* \))->H*, \( H \approx H^* \)
Canetti-Holmgren (ITCS16)
+ Zoom-in the core step:
++ Zoom-in the accumulator

Properties needed for the Accumulator
- Normal property like a Merkle-tree.
- **Enforcement (iO-friendly property):** there’s only one preimage $x^*$ of the current root value $y^*$.

Impossible information theoretically.

KLW’s **computational enforcement:**
Normal.Gen($\quad$)->H
Enforce.Gen( $x^*$, $y^*$)->H*, $H \approx H^*$

Alternatively: SSB hashing => [Ananth-Chen-Chung-Lin-Lin]
Selective Enforcing

Adaptive Enforcing
Selective Enforcing

$\mathbf{x^*} \leq \text{Adversary}$

Adaptive Enforcing
Selective Enforcing

\[ x^* \leq \text{Adversary} \]

\[ \text{Gen( )} \Rightarrow H \]

\[ \text{Enforcing}(x^*, y^*) \Rightarrow H^* \]

Adaptive Enforcing
Selective Enforcing

Gen() => H

x* <= Adversary

Adaptive Enforcing

Gen() => H

Enforcing(x*, y*) => H*

Gen() => H
Selective Enforcing

\[ x^* \leq \text{Adversary} \]

\[ \text{Gen}(\ ) \Rightarrow H \]

\[ \text{Enforcing}(x^*, y^*) \Rightarrow H^* \]

Adaptive Enforcing

\[ \text{Gen}(\ ) \Rightarrow H \]

\[ x^* \leq \text{Adversary}(H) \]
Selective Enforcing

Gen( ) => H

x* <= Adversary

Enforcing(x*, y*) => H*

Adaptive Enforcing

Gen( ) => H

x* <= Adversary(H)

Enforcing(x*, y*) => H*
Selective Enforcing

Gen( ) => H

Enforcing(x*, y*) => H*

Adaptive Enforcing

Gen( ) => H

x* <= Adversary(H)

Enforcing(x*, y*) => H*
#Mindblowing
Fact I
Can separate the key
key = hk + vk

Accumulator
iO-friendly Merkle-tree

Initialization
Authenticate
Update

G(D₀)
G(Pᵢ+1)
G(Dᵢ+1)

key

what is written in eprint 2015/1074
What is written in eprint 2015/1074

Accumulator
iO-friendly Merkle-tree

key = hk + vk

Setup(1^λ, S) samples Acc.PP ← Acc.Setup(1^λ, S) and samples a PPRF F.

For security, it is important that the accumulator is persistent and verifiable.

Setup(1^λ, S) = (sk, vk) ← Setup(1^λ, S(0, 0))

G(D_0)

key

hk

hk

Initialization

Authentication

Update

G(P_{i+1})

G(D_{i+1})

hk

vk

hk

vk

We note that M_i + can also be computed from C_i and Acc.PP. This means that later, when we prove security, it will suffice to analyze a game in which the adversary receives C_i instead of M_i. This also justifies our relatively informal description of M_i.
Adaptive Enforcing
Adaptive Enforcing

\[ x^* \leq \text{Adversary}(hk) \]
Adaptive Enforcing

\[ x^* \leq \text{Adversary}(hk) \]

\[ vk \approx vk^*(x^*) \]
Fact II
If you believe diO ...
key = hk + vk

Adaptive Enforcing
key = hk + vk

Adaptive Enforcing

always_hk_Gen(   ) -> hk := CRHF key h
key = hk + vk

Adaptive Enforcing

hk

x* <= Adversary(H)

always_hk_Gen( ) -> hk := CRHF key h
Adaptive Enforcing

key = hk + vk

\[ x^* \leq \text{Adversary}(H) \]

\[ \text{always}_\text{hk}_\text{Gen}() \rightarrow \text{hk} := \text{CRHF key} \ h \]

\[ \text{normal}_\text{vk}_\text{Gen}() \rightarrow \text{vk} \]
\[ \text{vk}(x,y) = \text{diO}( \text{if } h(x)=y, \text{output 1}; \text{else: output 0} \ ) \]
key = hk + vk

Adaptive Enforcing

\[ x^* \leq \text{Adversary}(H) \]

\[
\begin{align*}
\text{always}_h_k\_Gen( & ) \rightarrow hk := \text{CRHF key } h \\
\text{normal}_v_k\_Gen( & ) \rightarrow vk \\
& vk(x,y) = \text{diO( if } h(x)=y, \text{ output 1; else: output 0) }
\end{align*}
\]

\[
\begin{align*}
\text{enforce}_v_k\_Gen( x^*, y^* ) & \rightarrow vk^* \\
vk^*(x,y) & = \text{diO( if } y\neq y^* \text{ and } h(x)=y, \text{ output 1; Elseif } y=y^* \text{ and } x=x^*, \text{ output 1; Else: output 0) }
\end{align*}
\]
Fact III:
If you don’t believe diO, can still do this with iO.
From iO + preimage-bounded CRHF:

c-to-1 CRHF can be constructed from discrete-log or factoring
From iO + preimage-bounded CRHF:

c-to-1 CRHF can be constructed from discrete-log or factoring

\[ \text{enforce}_{vk}(x^*, y^*) \rightarrow vk^* \]

\[ vk^*(x,y) = \text{diO}(\text{if } y \neq y^* \text{ and } h(x) = y, \text{ output } 1; \]

\[ \text{Elseif } y = y^* \text{ and } x = x^*, \text{ output } 1; \]

\[ \text{Else: output } 0 \) \)
From iO + preimage-bounded CRHF:

c-to-1 CRHF can be constructed from discrete-log or factoring

\[
\text{enforce}_{vk}( x^*, y^* ) \rightarrow vk^*
\]

\[
vk^*(x,y) = \text{diO} ( \text{if } y\neq y^* \text{ and } h(x) = y, \text{ output } 1; \text{ Elseif } y=y^* \text{ and } x=x^*, \text{ output } 1; \text{ Else: output } 0 )
\]

By diO-iO equivalence lemma [Boyle-Chung-Pass ’14]:

“ If f1 and f2 differ only on polynomially many input-output values, and they are hard to find, then

\[
iO(f1) \approx iO(f2)
\]”
From iO + preimage-bounded CRHF:

c-to-1 CRHF can be constructed from discrete-log or factoring

\[ \text{enforce}_{vk}(x^*, y^*) \rightarrow vk^* \]
\[ vk^*(x,y) = \text{diO}( \text{if } y\neq y^* \text{ and } h(x)=y, \text{ output 1}; \]
\[ \text{Elseif } y=y^* \text{ and } x=x^*, \text{ output 1}; \]
\[ \text{Else: output 0) } \]

From shrinking 1 bit to length-halving: Merkle-Damgaard.
Fact IV: Adaptive Enforceable Accumulator done
Rest of the upgrades:

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

Indistinguishable as long as transc = (q, op) are the same. [KLW-technique. Assume iO]

q can be different [encrypt the state]

Memory content can be different [encrypt the data]

Hide access pattern. [oram]
Rest of the upgrades:

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

- Indistinguishable as long as transc = (q, op) are the same. [KLW-technique. Assume iO]
- q can be different [encrypt the state]
- Memory content can be different [encrypt the data]
- Hide access pattern. [oram]

+ adaptively enforceable accumulator [from iO+dlog or factoring]
Rest of the upgrades:

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

- Indistinguishable as long as transc = (q, op) are the same. [KLW-technique. Assume iO]
- q can be different [encrypt the state]
- Memory content can be different [encrypt the data]
- Hide access pattern. [oram]

+ adaptively enforceable accumulator [from iO+dlog or factoring]

Need a special property of the ORAM “Strong local randomness”, satisfied by Chung-Pass ORAM. With this property, can “guess” polynomially many addresses.
Rest of the upgrades:

Canetti-Holmgren scheme details:
Fixed-transcript => Fixed-access => Fixed-address => Fully secure

- Indistinguishable as long as $\text{transc} = (q, \text{op})$ are the same. [KLW-technique. Assume iO]
- $q$ can be different [encrypt the state]
- Memory content can be different [encrypt the data]
- Hide access pattern. [oram]

+ adaptively enforceable accumulator [from iO+dlog or factoring]

Need a special property of the ORAM “Strong local randomness”, satisfied by Chung-Pass ORAM. With this property, can “guess” polynomially many addresses.

[Ananth-Chen-Chung-Lin-Lin, eprint 2015/1082] can be viewed as accomplishing this for all the steps.
Summary

1. Adaptively secure garbled RAM with persistent memory.
2. Everything is succinct.
3. Upgrading to delegation with verifiability is almost for free.
4. “Reusability” is natural.
5. New iO-friendly tool: adaptively-enforceable accumulator (from iO+Preimage-bounded-CRHF)
Scenes
> sudo apt-get install GRAM_Canetti_Holmgren
> sudo apt-get install GRAM_Canetti_Holmgren
package indistinguishability_obfuscation not an accepted assumption, security at user’s own risk, continue (y) or not (n)
> sudo apt-get install GRAM_Canetti_Holmgren
package indistinguishability_obfuscation not an accepted assumption, security at user’s own risk, continue (y) or not (n)
y
> sudo apt-get install GRAM_Canetti_Holmgren
package indistinguishability_obfuscation not an accepted assumption, security at user’s own risk, continue (y) or not (n)
y
> upgrade GRAM_CCHR
Done
> sudo apt-get install GRAM_Canetti_Holmgren
package indistinguishability_obfuscation not an
accepted assumption, security at user’s own risk,
continue (y) or not (n)
y
> upgrade GRAM_CCHR
Done
> NSAcloud: GRAM_CCHR_Factoring RSA2048
> sudo apt-get install GRAM_Canetti_Holmgren
package indistinguishability_obfuscation not an accepted assumption, security at user’s own risk, continue (y) or not (n)
y
> upgrade GRAM_CCHR
Done

> NSAcloud: GRAM_CCHR_Factoring  RSA2048
Running time 1.0s
25195908475…20720357
= 83990...4079279 x 3091701...723883

Next question