Low-communication parallel quantum multi-target preimage search

Gustavo Banegas[1] and Daniel J. Bernstein[1,2]

Technische Universiteit **Eindhoven** University of Technology

August 18, 2017

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
gustavo@cryptme.in

[2] Department of Computer Science
University of Illinois at Chicago
djb@cr.yp.to

Introduction

Threat to AES:

- van Oorschot–Wiener "parallel rho method"

Introduction

Threat to AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.

Introduction

Threat to AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each running $2^{128}/pt$ fast steps, to find one of t independent AES keys $k_1, \ldots, k_t$, using a fixed plain text, e.g, AES(0).

Introduction

Threat to AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each running $2^{128}/pt$ fast steps, to find one of t independent AES keys $k_1, \ldots, k_t$, using a fixed plain text, e.g, AES(0).
- However, it is pre-quantum.

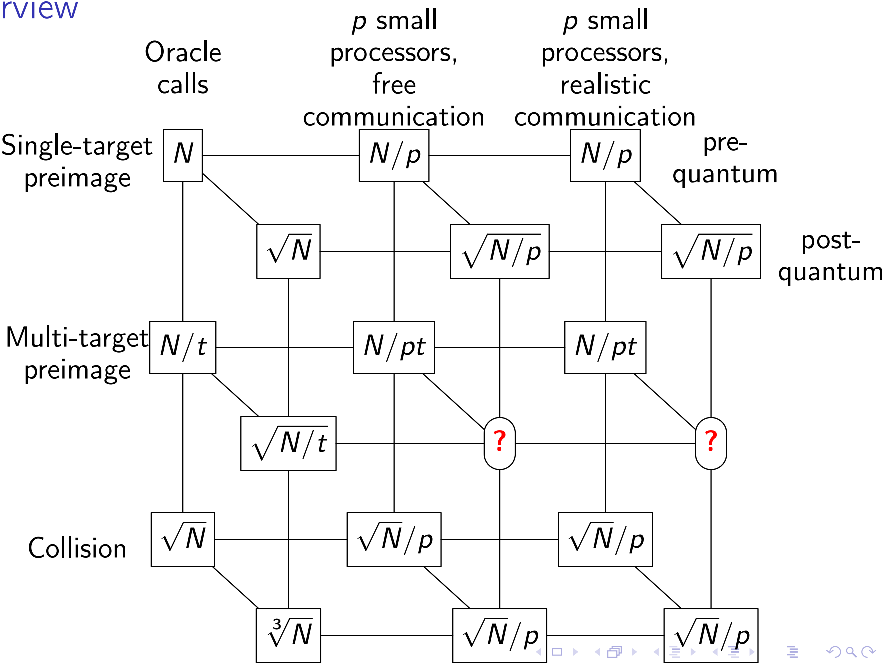
Introduction

Threat to AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each running $2^{128}/pt$ fast steps, to find one of t independent AES keys $k_1, \ldots, k_t$, using a fixed plain text, e.g, AES(0).
- However, it is pre-quantum.

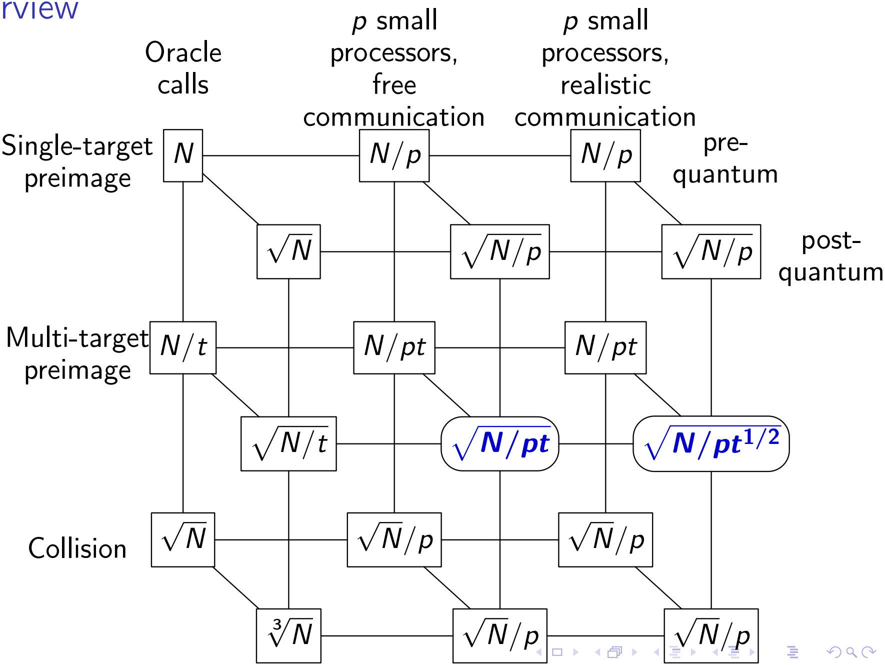
NIST has claimed that AES-128 is secure enough.

Overview



Oracle calls
p small processors, free communication
p small processors, realistic communication
pre-quantum
post-quantum
Single-target preimage
Multi-target preimage
Collision
N
N/p
N/p
$\sqrt{N}$
$\sqrt{N/p}$
$\sqrt{N/p}$
N/t
N/pt
N/pt
$\sqrt{N/t}$
?
?
$\sqrt{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$
$\sqrt[3]{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$

Overview


Oracle calls
p small processors, free communication
p small processors, realistic communication
Single-target preimage
Multi-target preimage
Collision
pre-quantum
post-quantum
N
N/p
N/p
√N
√(N/p)
√(N/p)
N/t
N/pt
N/pt
√(N/t)
√(N/pt)
√(N/pt^{1/2})
√N
√N/p
√N/p
∛N
√N/p
√N/p

Distinguish point

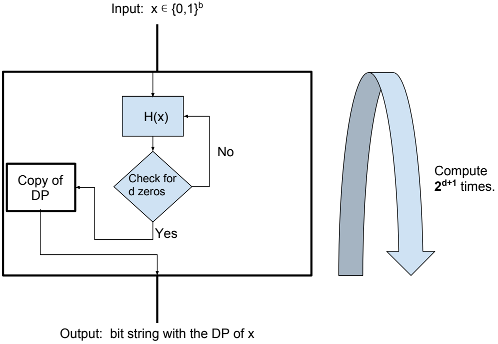
Let $H : \{0, 1\}^b$ to $\{0, 1\}^b$
Take x an input of H, $x' = H(x)$.
After take x' and apply H again, $x'' = H(x')$.
It is possible to do it n times, H^n until we satisfy a condition. In our case, we want the first $0 < d < b/2$ bits as 0.

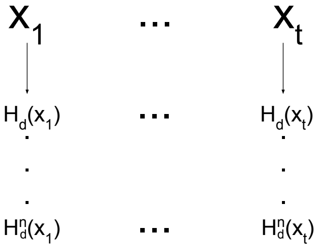
Distinguish point

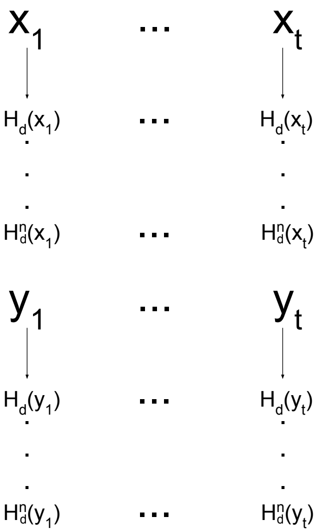
Let $H : \{0,1\}^b$ to $\{0,1\}^b$
Take x an input of H, $x' = H(x)$.
After take x' and apply H again, $x'' = H(x')$.
It is possible to do it n times, H^n until we satisfy a condition. In our case, we want the first $0 < d < b/2$ bits as 0.
$H_d^n(x)$ means d bits of x, computed n times.

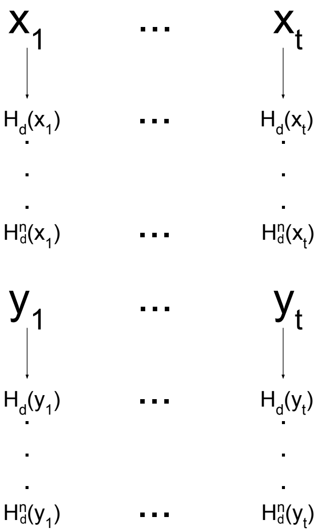
Distinguish point

Let $H : \{0,1\}^b$ to $\{0,1\}^b$
Take x an input of H, $x' = H(x)$.
After take x' and apply H again, $x'' = H(x')$.
It is possible to do it n times, H^n until we satisfy a condition. In our case, we want the first $0 < d < b/2$ bits as 0.
$H_d^n(x)$ means d bits of x, computed n times.




x_1
…
x_t
$H_d(x_1)$
…
$H_d(x_t)$
$H_d^n(x_1)$
…
$H_d^n(x_t)$


x1 ... xt
Hd(x1) ... Hd(xt)
Hnd(x1) ... Hnd(xt)
y1 ... yt
Hd(y1) ... Hd(yt)
Hnd(y1) ... Hnd(yt)



$$H_d^n(y_i) \stackrel{?}{=} H_d^n(x_j)$$

Reversibility

Reversibility of distinguish point

- Bennett-Tompa technique to build a reversible circuit for H^n;
- It is possible to achieve $a + O(b \log_2 n)$ ancillas and gate depth $O(gn^{1+\epsilon})$.

Reversibility of sorting on a mesh network

- Using the sorting strategy from "Efficient distributed quantum computing"[3];
- It is possible to perform the sorting of t elements using $O(t(b + (\log t)^2))$ ancillas and $O(t^{1/2}(\log t)^2)$ steps.

[3] Efficient distributed quantum computing
Beals, Robert and Brierley, Stephen and Gray, Oliver and Harrow, Aram W. and Kutin, Samuel and Linden, Noah and Shepherd, Dan and Stather, Mark

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.
- Precompute the chain ends for $y_1, \ldots, y_t$.

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.
- Precompute the chain ends for $y_1, \ldots, y_t$.
- Sort the chain ends for $x_1, \ldots, x_t$ and the chain ends for $y_1, \ldots, y_t$.

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.
- Precompute the chain ends for $y_1, \ldots, y_t$.
- Sort the chain ends for $x_1, \ldots, x_t$ and the chain ends for $y_1, \ldots, y_t$.
- If there is a collision, say a collision between the chain end for x_i and the chain end for y_j: recompute the chain for x_i, checking each chain element to see whether it is a preimage for y_j.

Finding t-images

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.
- Precompute the chain ends for $y_1, \ldots, y_t$.
- Sort the chain ends for $x_1, \ldots, x_t$ and the chain ends for $y_1, \ldots, y_t$.
- If there is a collision, say a collision between the chain end for x_i and the chain end for y_j: recompute the chain for x_i, checking each chain element to see whether it is a preimage for y_j.
- Output 0 if a preimage was found, otherwise 1.

Example

- Imagine a function $H : \{0,1\}^{40} \rightarrow \{0,1\}^{40}$;

Example

- Imagine a function $H : \{0,1\}^{40} \to \{0,1\}^{40}$;
- Let's say $t = 2^8$ and $p = 2^8$, for this example.

Example

- Imagine a function $H : \{0,1\}^{40} \to \{0,1\}^{40}$;
- Let's say $t = 2^8$ and $p = 2^8$, for this example.
- The probability to find one preimage is roughly $t^{5/2}/N = (2^8)^{5/2}/(2^{40}) \approx 2^{-20}$;
- Each processor is going to use $\sqrt{N/pt^{3/2}}$ iterations; $\sqrt{2^{40}/2^8((2^8)^{3/2})} = \sqrt{2^{40}/2^{20}} = 2^{10}$ iterations.
- Overall we get $(2^8)^{1/4}$ speedup from attacking 2^8 targets.

Example

- Imagine AES−128;

Example

- Imagine AES−128;
- Let's say $t = 2^{40}$ and $p = 2^{40}$, for this example.

Example

- Imagine AES−128;
- Let's say $t = 2^{40}$ and $p = 2^{40}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{40})^{5/2}/2^{128} \approx 2^{-28}$.

Example

- Imagine AES−128;
- Let's say $t = 2^{40}$ and $p = 2^{40}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{40})^{5/2}/2^{128} \approx 2^{-28}$.
- Each processor is going to use $\sqrt{N/pt^{3/2}}$ iterations;
- $\sqrt{2^{128}/2^{40}(2^{40})^{3/2}} \approx \sqrt{2^{128}/2^{100}}$

Example

- Imagine AES-128;
- Let's say $t = 2^{40}$ and $p = 2^{40}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{40})^{5/2}/2^{128} \approx 2^{-28}$.
- Each processor is going to use $\sqrt{N/pt^{3/2}}$ iterations;
- $\sqrt{2^{128}/2^{40}(2^{40})^{3/2}} \approx \sqrt{2^{128}/2^{100}}$
- $= \sqrt{2^{28}} = 2^{14}$ iterations.

Conclusion & What's next?

Conclusion:

- Circuit uses $O(a + tb + t(\log t)^2)$ ancillas;
- Depth of $O(\sqrt{N/pt^{1/2}}(gt^{\epsilon/2} + (\log t)^2 \log b))$;
- Approximately $\sqrt{N/pt^{3/2}}$ iterations.
- Created the circuit using quantum simulator for AES[4] (libquantum instead of LiQUi $|\rangle$);

[4] Applying Grover's algorithm to AES: quantum resource estimates Grassl, Markus and Langenberg, Brandon and Roetteler, Martin and Steinwandt, Rainer

Conclusion & What's next?

Conclusion:

- Circuit uses $O(a + tb + t(\log t)^2)$ ancillas;
- Depth of $O(\sqrt{N/pt^{1/2}}(gt^{\epsilon/2} + (\log t)^2 \log b))$;
- Approximately $\sqrt{N/pt^{3/2}}$ iterations.
- Created the circuit using quantum simulator for AES[4] (libquantum instead of LiQ$Ui|\rangle$);

What's next?

- Check for the real number of qubits/gates;
- Is it possible to improve?

[4] Applying Grover's algorithm to AES: quantum resource estimates Grassl, Markus and Langenberg, Brandon and Roetteler, Martin and Steinwandt, Rainer

Questions

Thank you for your attention.
Questions?
gustavo@cryptme.in