Preimage search using low communication cost parallel Grover algorithm

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

Technische Universiteit
Eindhoven
University of Technology

Crypto Working Group
September 8, 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

Preimage

Let H be a function that $H : \{0,1\}^n \rightarrow \{0,1\}^n$. Preimage search is given an output y, find a x such that $H(x) = y$.

Introduction

Preimage

Let H be a function that $H : \{0,1\}^n \rightarrow \{0,1\}^n$. Preimage search is given an output y, find a x such that $H(x) = y$.
It is desirable that given an output it should be computationally infeasible to find any input that hashes to that output.

Introduction

Preimage

Consider $n = 128$ and $H = \text{AES}$ and 0 fixed as a plain text, i.e., $H(x) = AES_x(0)$, where x is a key.

Introduction

Preimage

Consider $n = 128$ and $H = \mathrm{AES}$ and 0 fixed as a plain text, i.e., $H(x) = AES_x(0)$, where x is a key.
The complexity to find one key is 2^{128} guesses.

Introduction

Brute-force search for one preimage

Let H be a function that $H : \{0,1\}^n \rightarrow \{0,1\}^n$.
The brute force is to check every input x given an output y. The time complexity will be 2^n guesses using classical computers.

Introduction

Brute-force search for one preimage

Let H be a function that $H : \{0,1\}^n \rightarrow \{0,1\}^n$.
The brute force is to check every input x given an output y. The time complexity will be 2^n guesses using classical computers.
If we apply Grover's algorithm , using a quantum computer, the complexity decreases to $2^{n/2}$ guesses.

Introduction

Brute-force search for multi target preimages

Let H be a function that $H : \{0,1\}^n \to \{0,1\}^n$.
However, we have a set of output y's, i.e., $Y = \{y_1, y_2, \ldots, y_t\}$ and we want to find one y_i.

Introduction

Brute-force search for multi target preimages

Let H be a function that $H : \{0,1\}^n \rightarrow \{0,1\}^n$.
However, we have a set of output y's, i.e., $Y = \{y_1, y_2, \ldots, y_t\}$ and we want to find one y_i.
Now, we verify every input x with set of output Y. If we **ignore several costs**, the complexity decreases to $2^n/t$ guesses in a classical computer.
If we apply Grover's algorithm, using a quantum computer, the complexity decreases to $2^{n/2}/t^{1/2}$ guesses.

Introduction

Costs for comparison

One big cost for preimage search in both cases is the comparisons.

Introduction

Costs for comparison

One big cost for preimage search in both cases is the comparisons.

- Classical computer:
  - Single target: (2^n)
  - Multi target: $t * 2^n/t$

Introduction

Costs for comparison

One big cost for preimage search in both cases is the comparisons.

- Classical computer:
  - Single target: (2^n)
  - Multi target: $t * 2^n/t$
- Quantum computer:
  - Single target: $2^{n/2}$
  - Multi target: $t * 2^{n/2}/t^{1/2}$

Introduction

Parallel multi-target image attack for AES:

- van Oorschot–Wiener “parallel rho method”

Introduction

Parallel multi-target image attack for AES:

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

Introduction

Parallel multi-target image attack for AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each processor runs $2^{128}/pt$ 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

Parallel multi-target image attack for AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each processor runs $2^{128}/pt$ 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

Parallel multi-target image attack for AES:

- van Oorschot–Wiener "parallel rho method"
  - It uses a mesh of p small processors.
  - Each processor runs $2^{128}/pt$ 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.

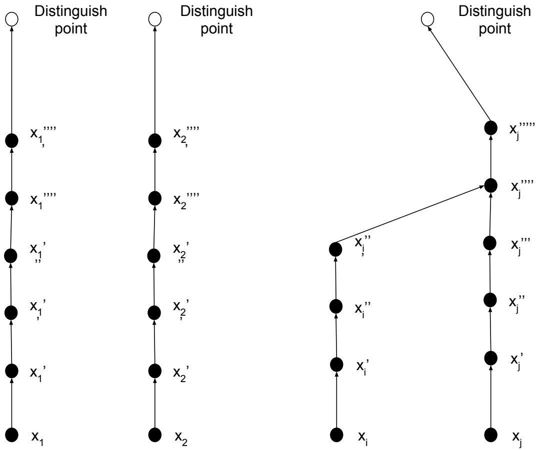
Introduction - Parallel rho method

Distinguish Point

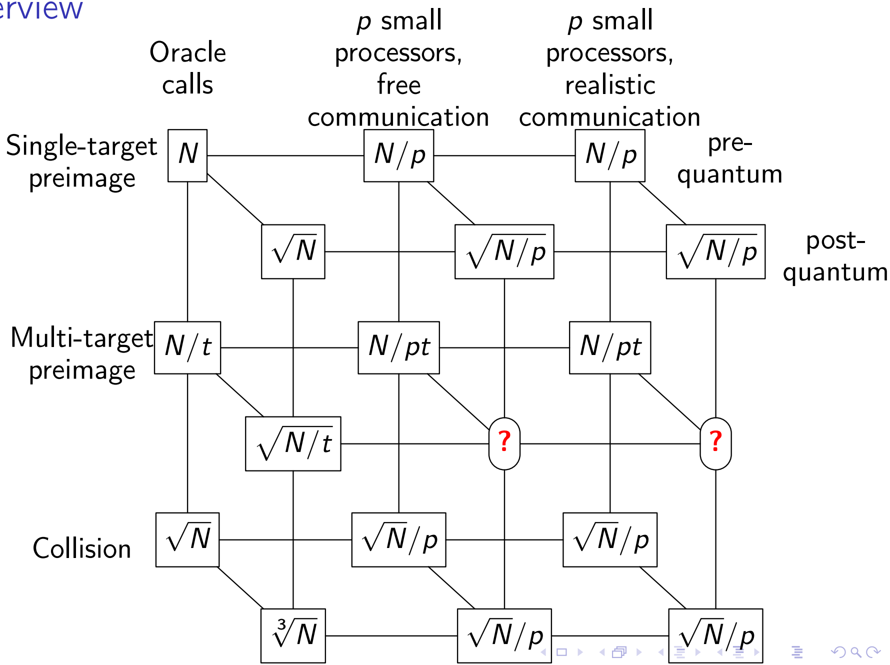
Consider $H : \{0,1\}^b \to \{0,1\}^b$
Take x an input of H, $x' = H(x)$.
Thereafter, take x' and apply H again, $x'' = H(x')$.
It is possible to do it n times (H^n), until a given condition is satified. In our case, we want the first $0 < d < b/2$ bits as 0.
$H_d^n(x)$ means d bits of x, computed n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

Introduction - Parallel rho method

Distinguish Point

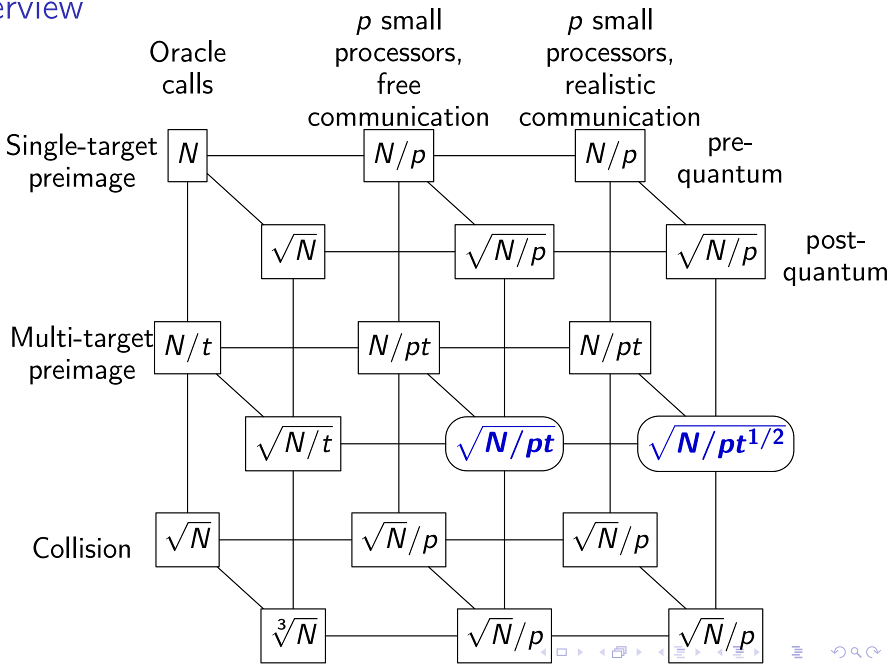


Overview



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
$\sqrt{N}$
$\sqrt{N/p}$
$\sqrt{N/p}$
N/t
N/pt
N/pt
$\sqrt{N/t}$
$\sqrt{N/pt}$
$\sqrt{N/pt^{1/2}}$
$\sqrt{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$
$\sqrt[3]{N}$
$\sqrt{N}/p$
$\sqrt{N}/p$

Distinguish point in quantum setting

Distinguish point in quantum computers

- The operations in quantum computer must be **reversible**;

Distinguish point in quantum setting

Distinguish point in quantum computers

- The operations in quantum computer must be **reversible**;
- It is not possible to design a "simple circuit" for distinguish point;

Distinguish point in quantum setting

Distinguish point in quantum computers

- The operations in quantum computer must be **reversible**;
- It is not possible to design a “simple circuit” for distinguish point;
- The sorting needs to be reversible too.

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

time 0: x y

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

time 0:	x	y	
time 1:	x	y	$H(x)$

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

time 0:	x	y	
time 1:	x	y	$H(x)$
time 2:	x	y	$H^2(x)$

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

time 0:	x	y	
time 1:	x	y	$H(x)$
time 2:	x	y	$H^2(x)$
time 3:	x	$y + H^3(x)$	

Distinguish point in quantum setting

Using classical computers

Example to compute $H^3(x)$:

time 0:	x	y	
time 1:	x	y	$H(x)$
time 2:	x	y	$H^2(x)$
time 3:	x	$y + H^3(x)$	

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0
time 3:	x	$y + H^3(x)$	$H(x)$	$H^2(x)$	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0
time 3:	x	$y + H^3(x)$	$H(x)$	$H^2(x)$	0
time 4:	x	$y + H^3(x)$	$H(x)$	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

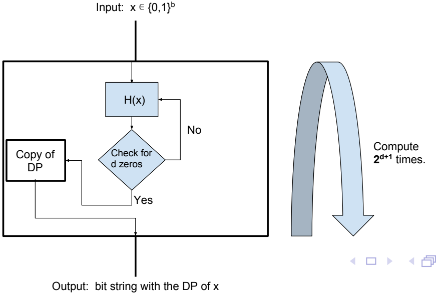
time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0
time 3:	x	$y + H^3(x)$	$H(x)$	$H^2(x)$	0
time 4:	x	$y + H^3(x)$	$H(x)$	0	0
time 5:	x	$y + H^3(x)$	0	0	0

Distinguish point in quantum setting

Trade-off from Bennett–Tompa

Example to compute $H^3(x)$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0
time 3:	x	$y + H^3(x)$	$H(x)$	$H^2(x)$	0
time 4:	x	$y + H^3(x)$	$H(x)$	0	0
time 5:	x	$y + H^3(x)$	0	0	0



$$
\begin{array}{ccc}
x_1 & \cdots & x_t \\
\downarrow & & \downarrow \\
H_d(x_1) & \cdots & H_d(x_t) \\
\vdots & & \vdots \\
H_d^n(x_1) & \cdots & H_d^n(x_t)
\end{array}
$$

$$\begin{array}{ccc} x_1 & \cdots & x_t \\ \downarrow & & \downarrow \\ H_d(x_1) & \cdots & H_d(x_t) \\ \vdots & & \vdots \\ H_d^n(x_1) & \cdots & H_d^n(x_t) \end{array}$$

$$\begin{array}{ccc} y_1 & \cdots & y_t \\ \downarrow & & \downarrow \\ H_d(y_1) & \cdots & H_d(y_t) \\ \vdots & & \vdots \\ H_d^n(y_1) & \cdots & H_d^n(y_t) \end{array}$$

$$
\begin{array}{ccc}
x_1 & \cdots & x_t \\
\downarrow & & \downarrow \\
H_d(x_1) & \cdots & H_d(x_t) \\
\vdots & & \vdots \\
H_d^n(x_1) & \cdots & H_d^n(x_t) \\
 & & \\
y_1 & \cdots & y_t \\
\downarrow & & \downarrow \\
H_d(y_1) & \cdots & H_d(y_t) \\
\vdots & & \vdots \\
H_d^n(y_1) & \cdots & H_d^n(y_t)
\end{array}
$$

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

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 $O(gn^{1+\epsilon})$.

[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

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];
- We used Odd-even mergesort;
- 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 $H : \{0,1\}^{40} \to \{0,1\}^{40}$;
- Consider $t = 2^8$ and $p = 2^8$, for this example.

Example

- Imagine a function $H : \{0,1\}^{40} \rightarrow \{0,1\}^{40}$;
- Consider $t = 2^8$ 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;
- Consider $t = 2^{30}$ and $p = 2^{30}$, for this example.

Example

- Imagine AES$-$128;
- Consider $t = 2^{30}$ and $p = 2^{30}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{30})^{5/2}/2^{128} \approx 2^{-53}$.

Example

- Imagine AES-128;
- Consider $t = 2^{30}$ and $p = 2^{30}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{30})^{5/2}/2^{128} \approx 2^{-53}$.
- Each processor is going to use $\sqrt{N/pt^{3/2}}$ iterations;
- $\sqrt{2^{128}/2^{30}(2^{30})^{3/2}} \approx \sqrt{2^{128}/2^{75}}$

Example

- Imagine AES$-$128;
- Consider $t = 2^{30}$ and $p = 2^{30}$, for this example.
- The probability to find is roughly $t^{5/2}/N$; For our example: $(2^{30})^{5/2}/2^{128} \approx 2^{-53}$.
- Each processor is going to use $\sqrt{N/pt^{3/2}}$ iterations;
- $\sqrt{2^{128}/2^{30}(2^{30})^{3/2}} \approx \sqrt{2^{128}/2^{75}}$
- $= \sqrt{2^{53}} \approx 2^{26}$ 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 LiQ$Ui|\rangle$);

[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