Faster Algorithms For Approximate Common Divisors:

Breaking FHE Challenges Over The Integers

Yuanmi Chen[1], Phong Q. Nguyen[2]

Eurocrypt, 2012

[1] ENS (France)
[2] INRIA (France) & Tsinghua University (China)

Motivation: Security of FHE schemes

Fully Homomorphic Encryption: Perform calculation on encrypted data!

There are many FHE schemes now, but very few come with **concrete parameters**:

1. Ideal lattices: [Gen09] $\longrightarrow$[GH10].
2. Approximate GCD: [vDGHV10] $\longrightarrow$[CMNT11].
3. . . .

Question: How **secure** are the systems?

FHE based on Approx-GCD

We studies [vDGHV10] and [CMNT11] schemes, which are based on Approx-GCD problem, introduced in [HG01].

[CMNT11] schemes proposed unusual parameters, e.g. huge numbers.

It is not clear what is the best attack.

Our work

1. A time/memory trade-off for "algebraic" exhaustive search:

   Improved attack on the FHE scheme of [CMNT11].

2. Several applications to cryptanalysis:

   RSA, Factoring.

Note: a similar trade-off already appeared in Strassen factoring algorithm, but we show new uses.

Approximate Common-Divisor Problem

Problem:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 + r_1 \\ x_2 &= pq_2 + r_2 \\ &\dots \end{aligned}$$

where p is a secret big prime. We are given x_i's which are near multiples of p, except x_0.

Question: Given x_i, recover p?

Greatest Common-Divisor Problem

If there were no noise, the problem would be easy:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 \end{aligned}$$

where p is a secret big prime. We are given x_i's which are exact multiples of p.

Question: Given x_0, x_1, recover p?

Greatest Common-Divisor Problem

If there were no noise, the problem would be easy:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 \end{aligned}$$

where p is a secret big prime. We are given x_i's which are exact multiples of p.

Question: Given x_0, x_1, recover p?

Answer: $p \leftarrow gcd(x_0, x_1)$.

Approximate Common-Divisor Problem

Problem:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 + r_1 \\ &\dots \end{aligned}$$

where p is a secret big prime. We are given x_i's which are near multiples of p, except x_0

Question: Given x_i, and suppose $r_i < 2^\rho$, recover p?

Approximate Common-Divisor Problem

Problem:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 + r_1 \\ &\dots \end{aligned}$$

where p is a secret big prime. We are given x_i's which are near multiples of p, except x_0

Question: Given x_i, and suppose $r_i < 2^\rho$, recover p?

Enumerate over all possible r.

$$p \leftarrow \gcd(x_0, (x_1 - i) \mod x_0) \qquad \forall 0 \leqslant i \leqslant 2^\rho - 1$$

2^ρ GCD operations.

Approximate Common-Divisor Problem

Problem:

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 + r_1 \\ &\dots \end{aligned}$$

where p is a secret big prime. We are given x_i's which are near multiples of p, except x_0

Question: Given x_i, and suppose $r_i < 2^\rho$, recover p?

Enumerate over all possible r.

$$p \leftarrow \gcd\left(x_0, \prod_{i=0}^{2^\rho - 1}(x_1 - i) \bmod x_0\right)$$

2^ρ multiplications. ≈ 5 times faster than GCD

Our Algorithm

Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

Our Algorithm

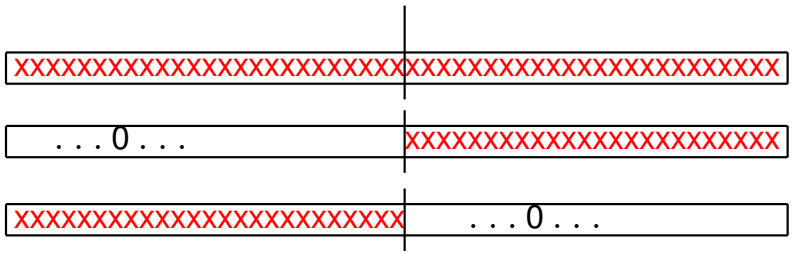
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



Our Algorithm

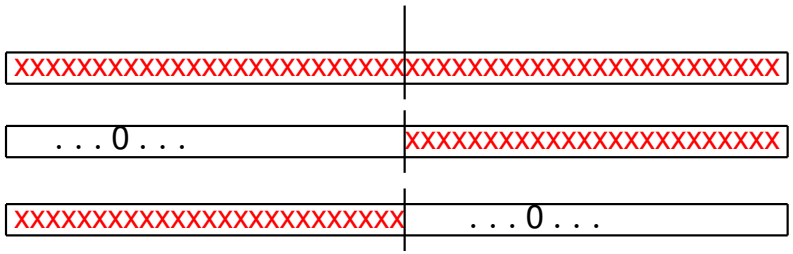
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

`forall` $s_{i,j}$

$\quad p \leftarrow p \cdot (x_1 - s_{i,j})$

Our Algorithm

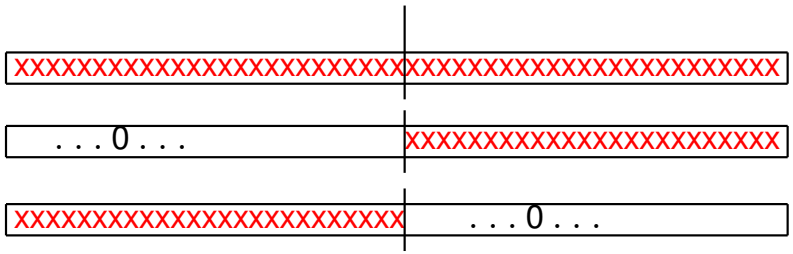
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

`forall` $s_{i,j}$ $\quad \|S\| = 2^\rho$

$\quad p \leftarrow p \cdot (x_1 - s_{i,j})$

Our Algorithm

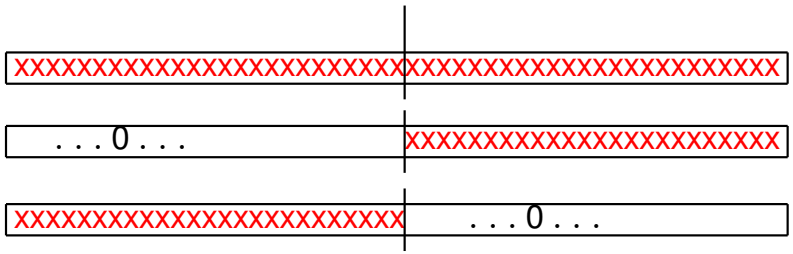
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$
forall u_i $\quad \|S\| = 2^\rho$
  forall v_j $\quad (x_1 - s_{i,j})$
    $p \leftarrow p \cdot (x_1 - u_i - v_j)$

Our Algorithm

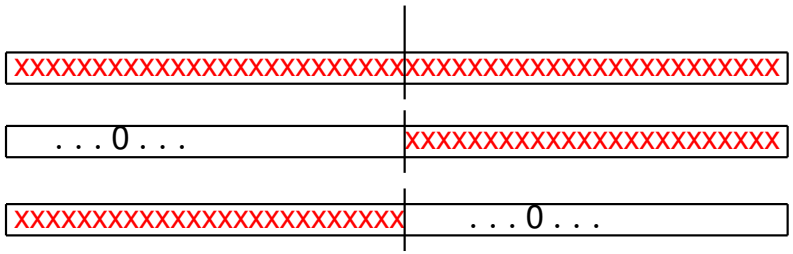
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

forall u_i

forall v_j

$p \leftarrow p \cdot (x_1 - u_i - v_j)$

Our Algorithm

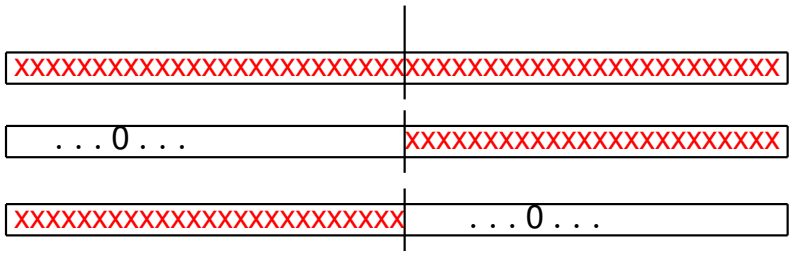
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

forall u_i

Calculate in time $\tilde{O}(2^{\rho/2})$

$F(x) \leftarrow \prod_j (x_1 - x - v_j)$

$p \leftarrow p \cdot F(u_i)$

Our Algorithm

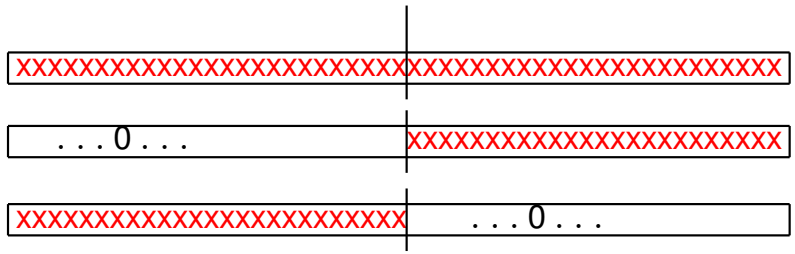
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

$F(x) \leftarrow \prod_j (x_1 - x - v_j)$ — polynomial degree $2^{\rho/2}$

forall u_i — $2^{\rho/2}$ points

$p \leftarrow p \cdot F(u_i)$

Our Algorithm

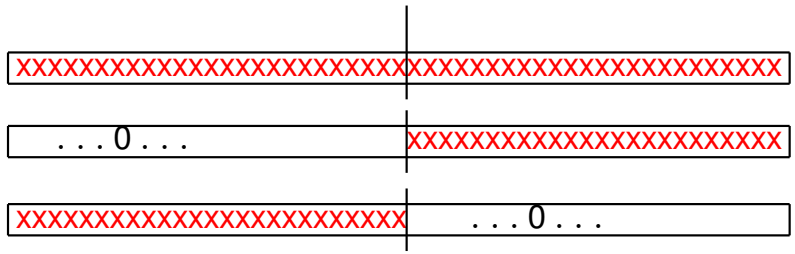
Goal: $\prod_{i=0}^{2^\rho-1}(x_1 - i) \mod x_0$

Observation: The set $S = \{0, \ldots, 2^\rho - 1\}$ is structured:

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$p \leftarrow 1$

forall u_i

$F(x) \leftarrow \prod_j (x_1 - x - v_j)$ — polynomial degree $2^{\rho/2}$

$2^{\rho/2}$ points

$p \leftarrow p \cdot F(u_i)$

Classical algorithms for fast evaluation of a $2^{\rho/2}$-degree polynomial at $2^{\rho/2}$ points in $\tilde{O}(2^{\rho/2})$ time and space

Classical Tools

One knows how to:

1. Construct the polynomial $\prod_{i=0}^{2^s-1}(X - a_i)$ using **product tree** in time $\tilde{O}(2^s)$.

Classical Tools

One knows how to:

1. Construct the polynomial $\prod_{i=0}^{2^s-1}(X - a_i)$ using **product tree** in time $\tilde{O}(2^s)$.
2. **Evaluate** $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Classical Tools

One knows how to:

1. Construct the polynomial $\prod_{i=0}^{2^s-1}(X-a_i)$ using **product tree**

   in time $\tilde{O}(2^s)$.
2. **Evaluate** $f(X)$ on $\{a_0,\ldots,a_{2^s-1}\}$, $\deg(f)\leqslant 2^s$
   in time $\tilde{O}(2^s)$.

   Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X-a_i)$. $\tilde{O}(2^s)$

Classical Tools

One knows how to:

1. Construct the polynomial $\prod_{i=0}^{2^s-1}(X-a_i)$ using **product tree** in time $\tilde{O}(2^s)$.
2. **Evaluate** $f(X)$ on $\{a_0,\ldots,a_{2^s-1}\}$, $\deg(f)\leqslant 2^s$ in time $\tilde{O}(2^s)$.

   Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X-a_i)$. $\tilde{O}(2^s)$

   Step 2 Calculate $f(a_i)\leftarrow f(X) mod (X-a_i), \forall i$. $\tilde{O}(2^s)$

Tool (1): Product Tree

Calculate $\prod_{i=0}^{2^s-1}(X - a_i)$ in time $\tilde{O}(2^s)$.

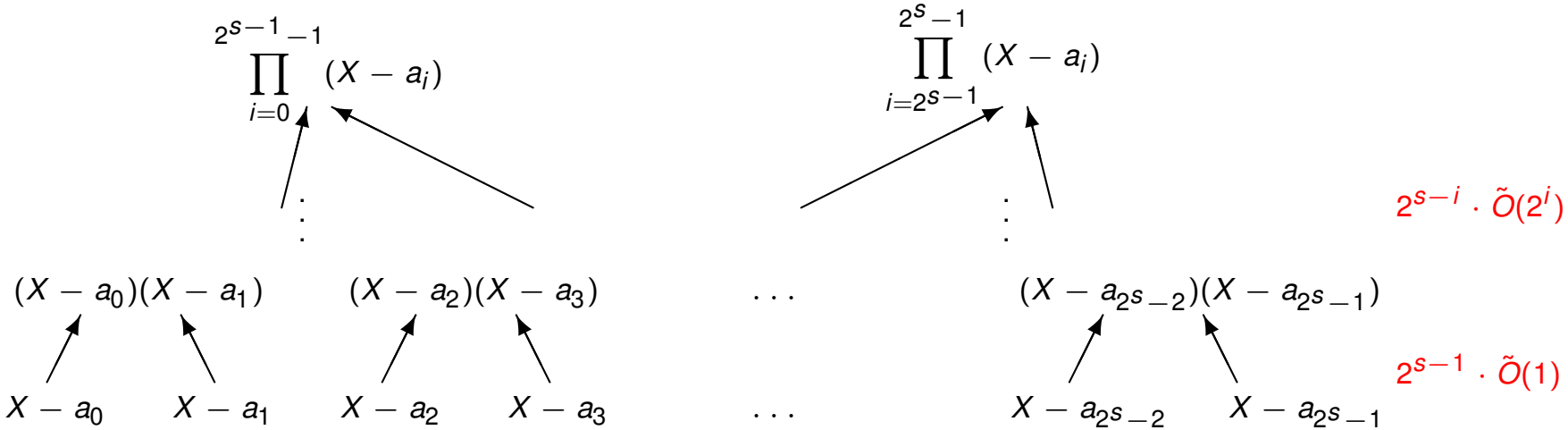
$X - a_0$ $X - a_1$ $X - a_2$ $X - a_3$ $\ldots$ $X - a_{2^s-2}$ $X - a_{2^s-1}$

Tool (1): Product Tree

Calculate $\prod_{i=0}^{2^s-1}(X - a_i)$ in time $\tilde{O}(2^s)$.

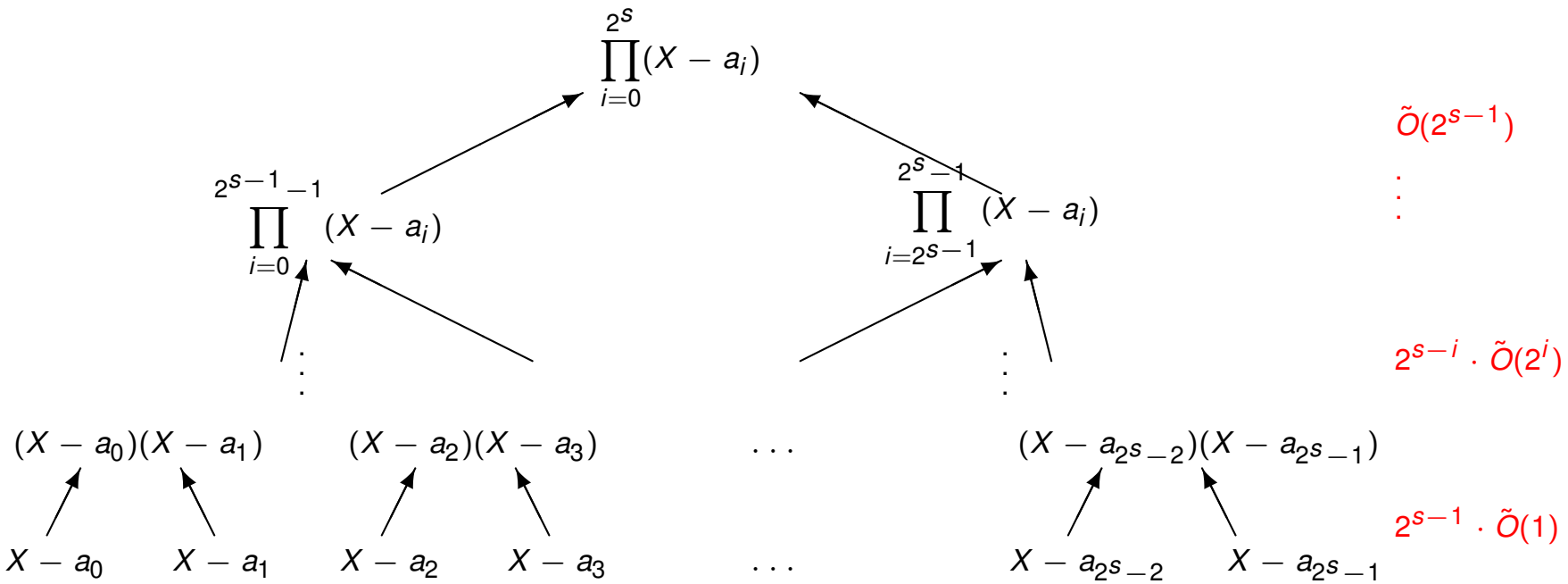
Tool (1): Product Tree

Calculate $\prod_{i=0}^{2^s-1}(X-a_i)$ in time $\tilde{O}(2^s)$.



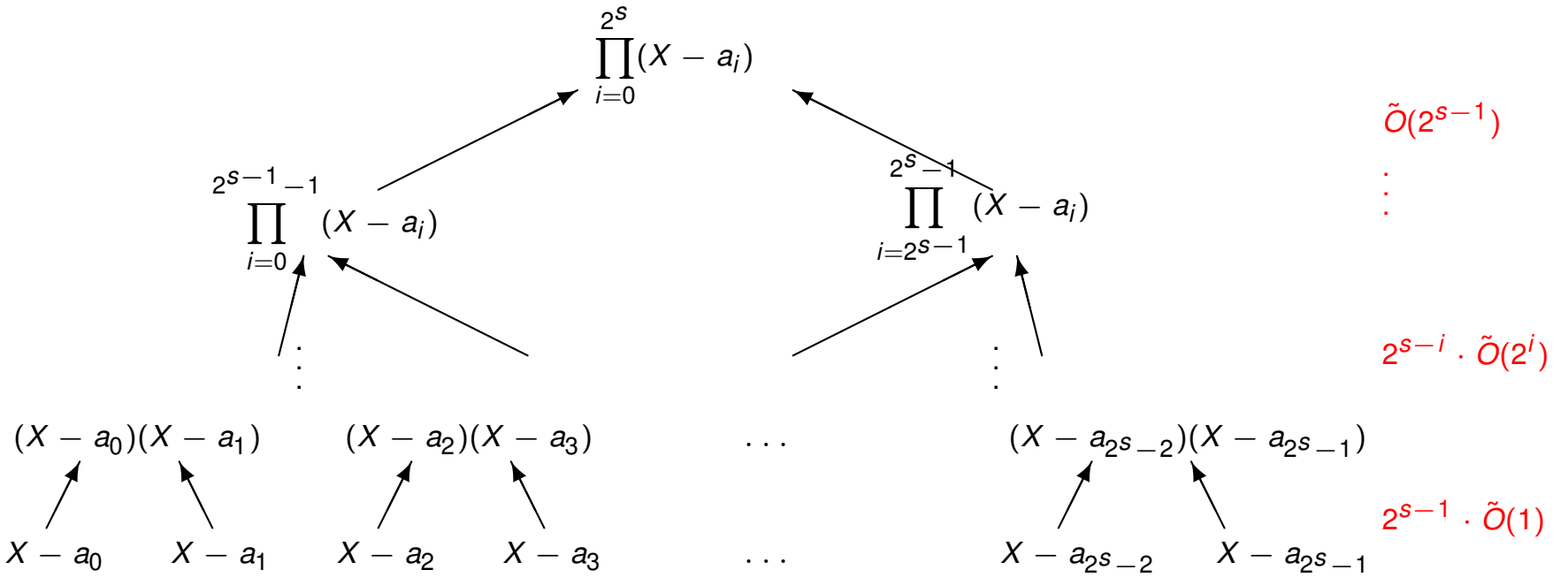
Tool (1): Product Tree

Calculate $\prod_{i=0}^{2^s-1}(X-a_i)$ in time $\tilde{O}(2^s)$.



Tool (1): Product Tree

Calculate $\prod_{i=0}^{2^s-1}(X-a_i)$ in time $\tilde{O}(2^s)$.



Total Time Complexity: $\tilde{O}(2^s)$

Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

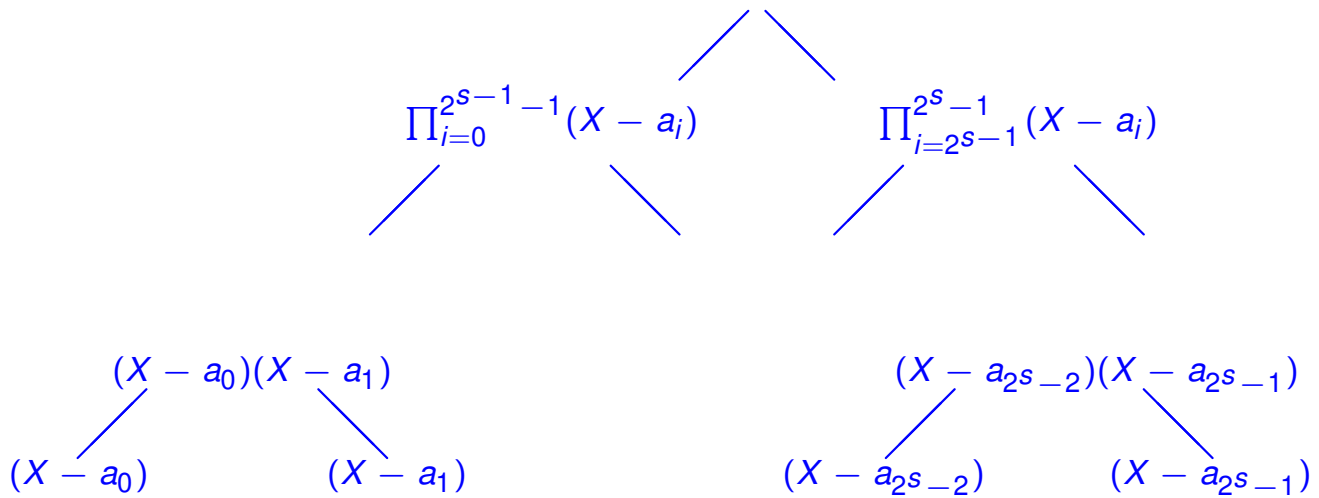
Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

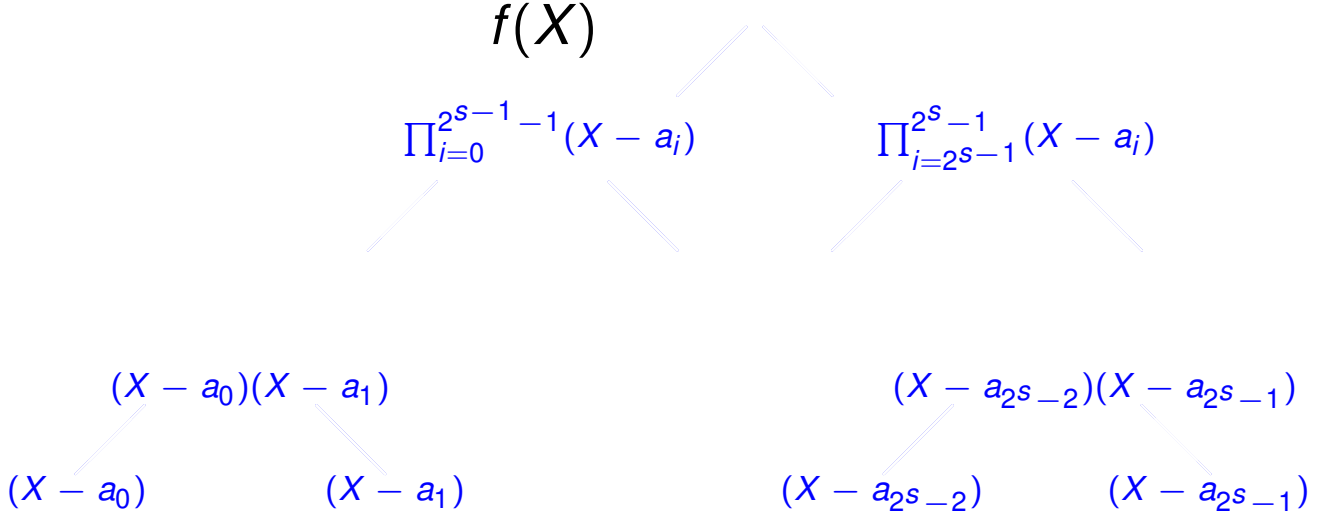


Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

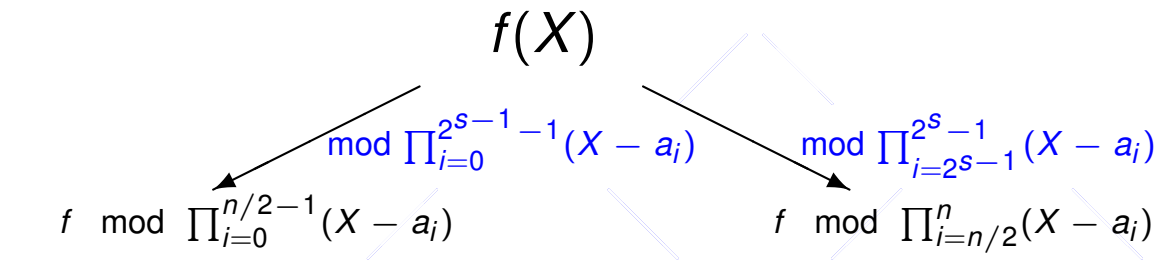


Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$



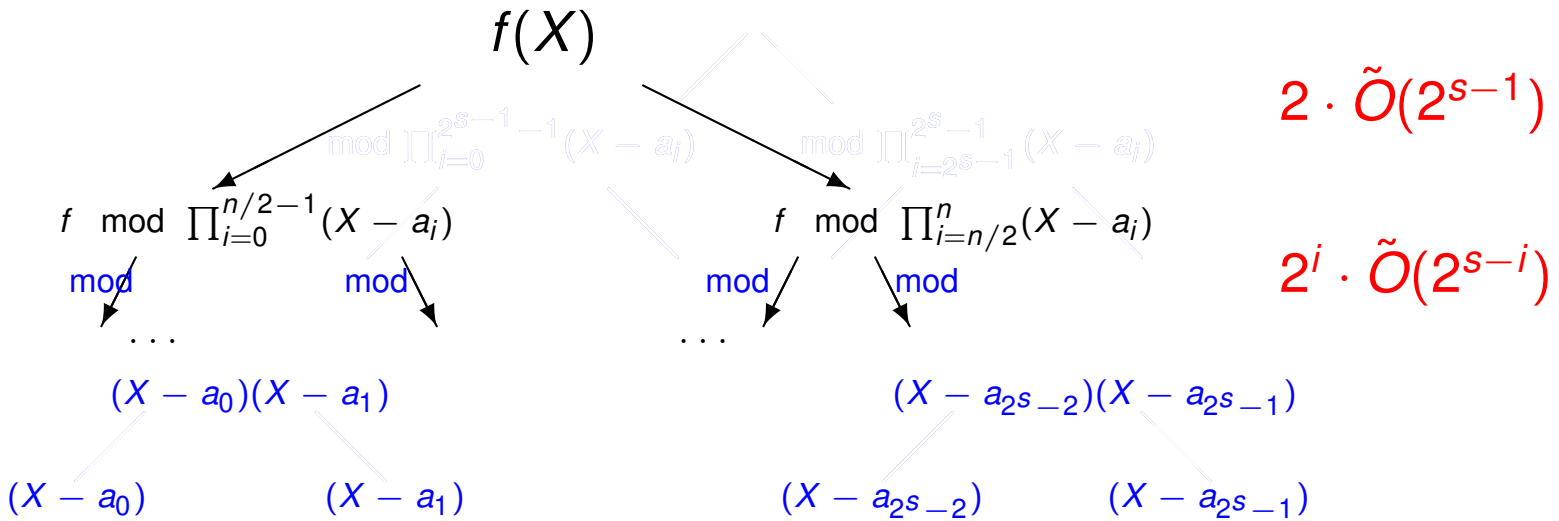
$2 \cdot \tilde{O}(2^{s-1})$

Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

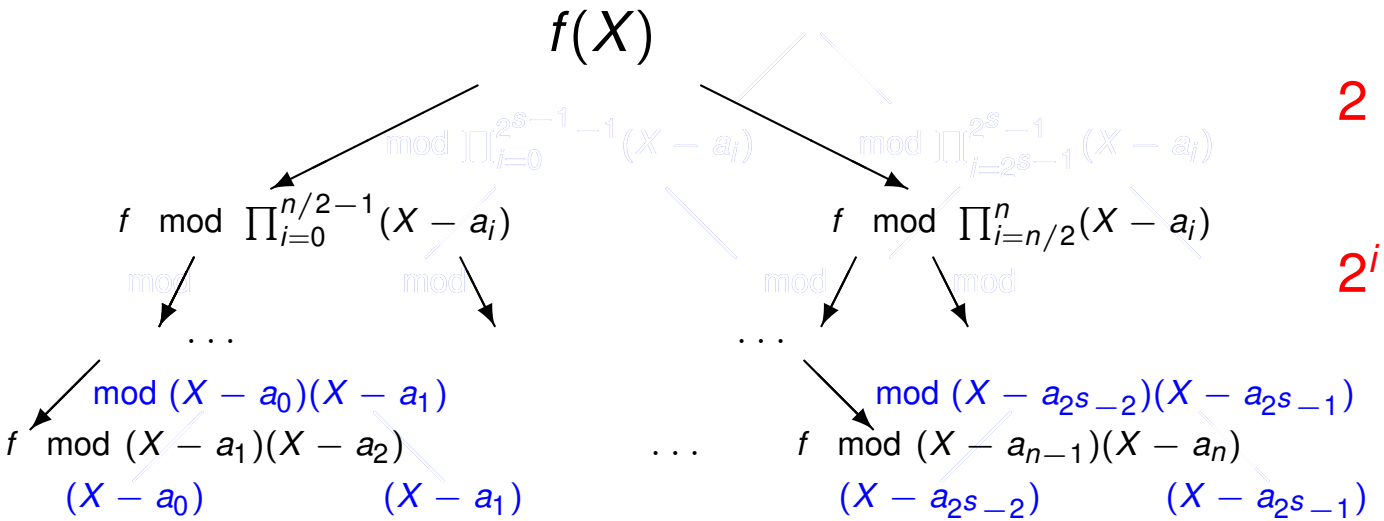


Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

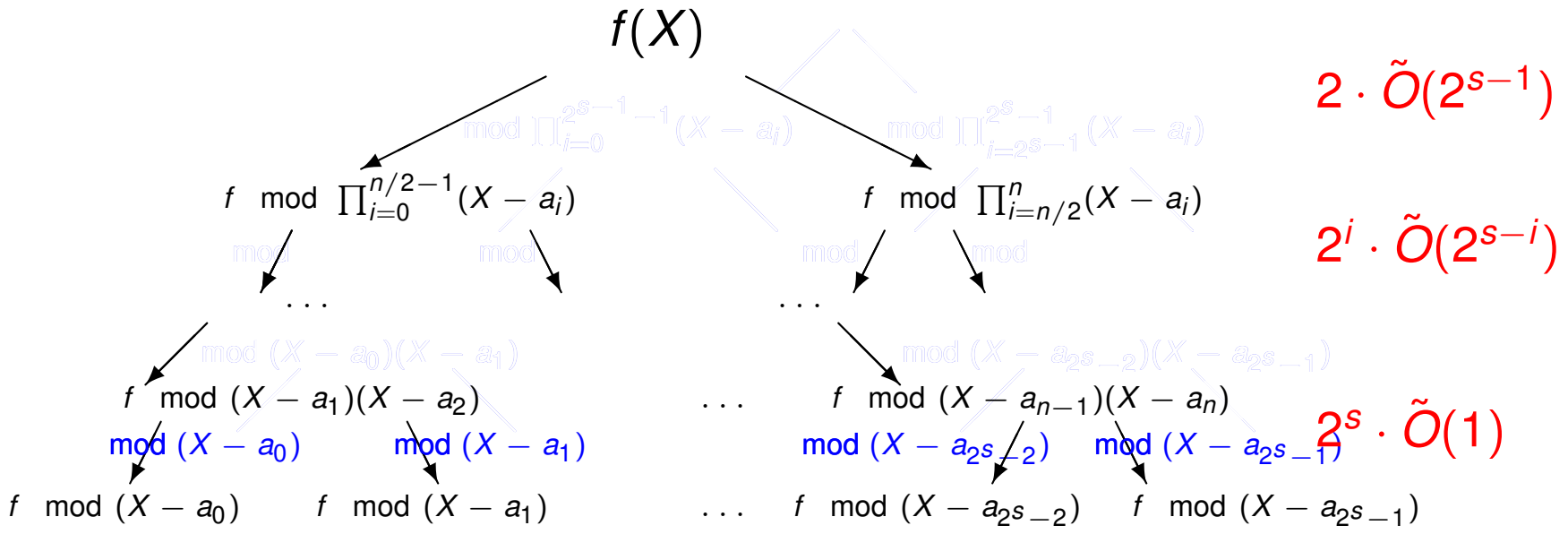


Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

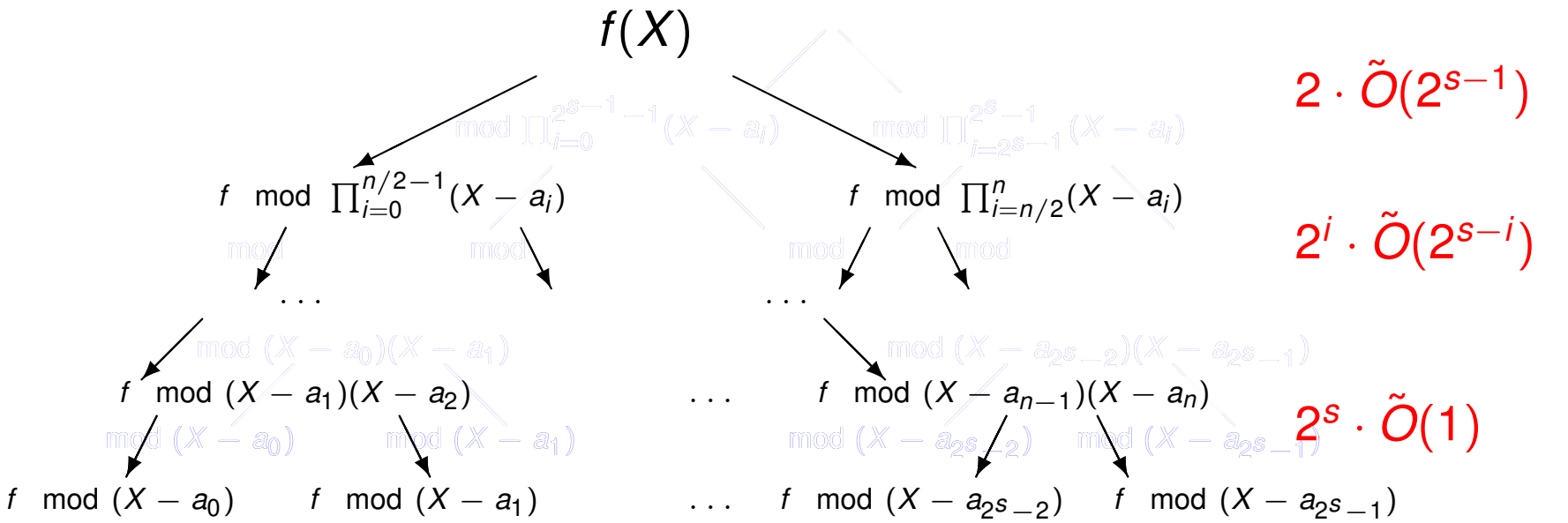


Tool (2): Evaluation

Evaluate $f(X)$ on $\{a_0, \ldots, a_{2^s-1}\}$, $\deg(f) \leqslant 2^s$ in time $\tilde{O}(2^s)$.

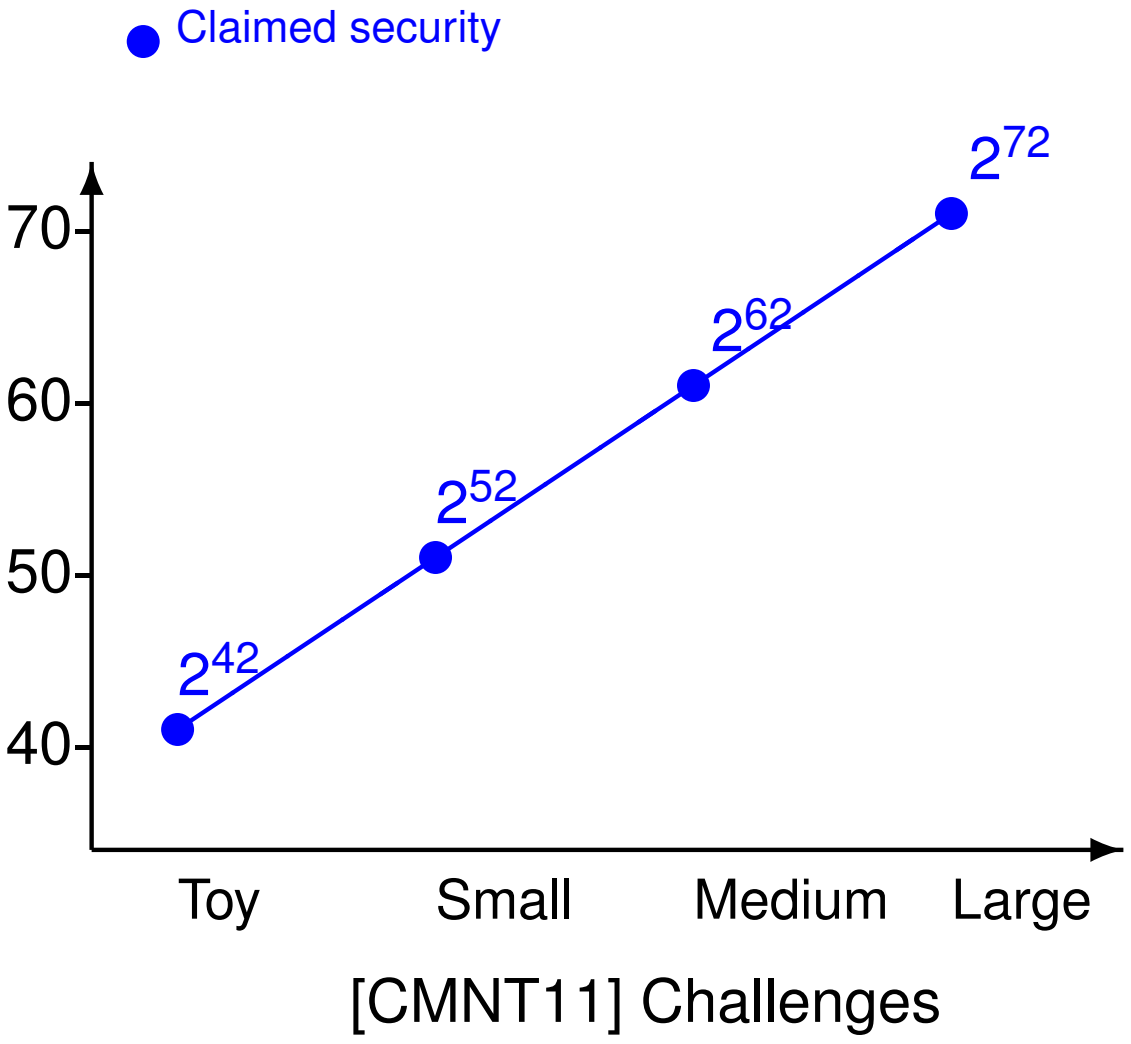
Step 1 Construct product tree $\prod_{i=0}^{2^s-1}(X - a_i)$. $T = \tilde{O}(2^s)$

Step 2 Calculate $f(a_i) \leftarrow f(X) mod (X - a_i)$

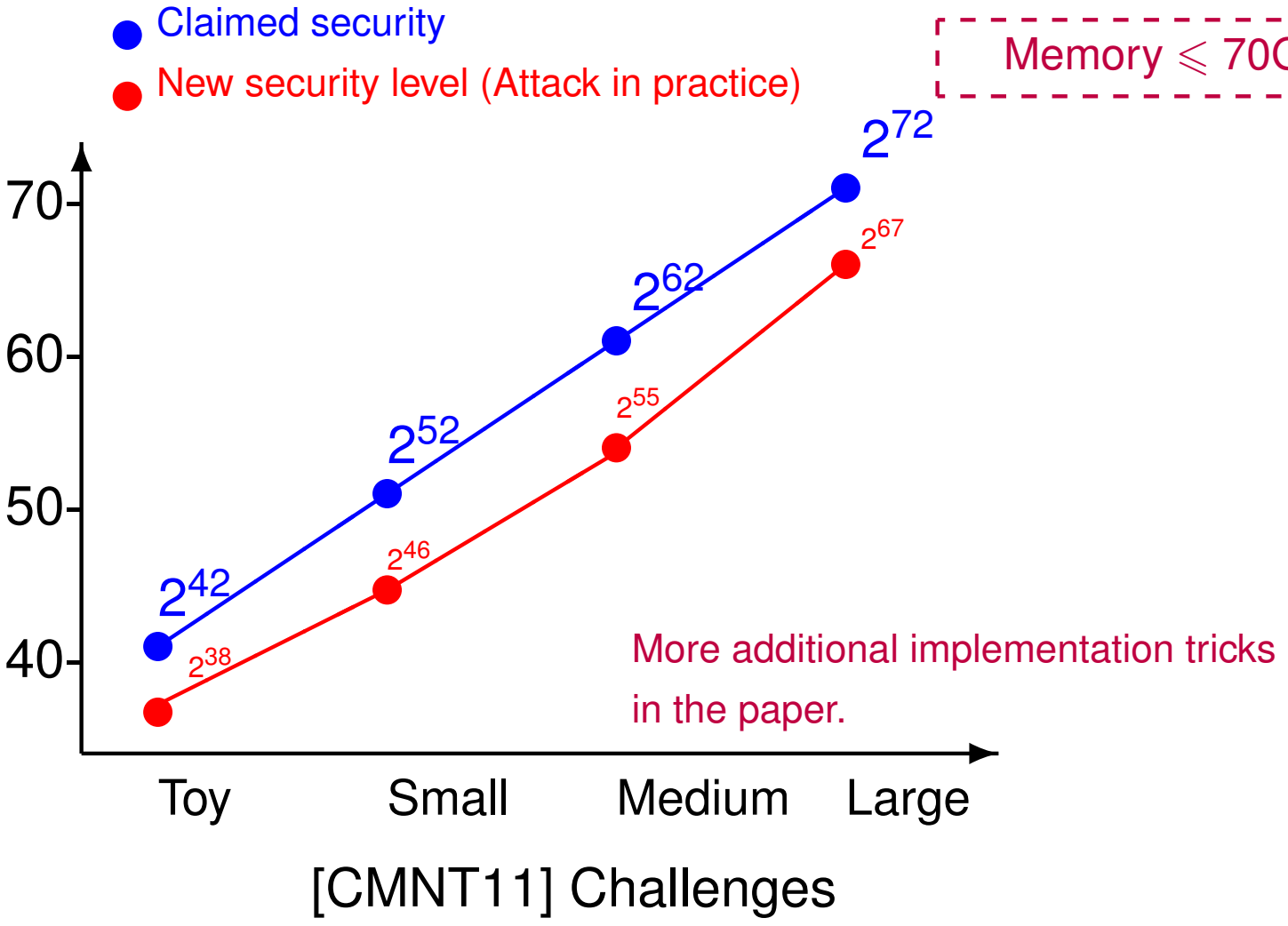


Total Time Complexity: $\tilde{O}(2^s)$

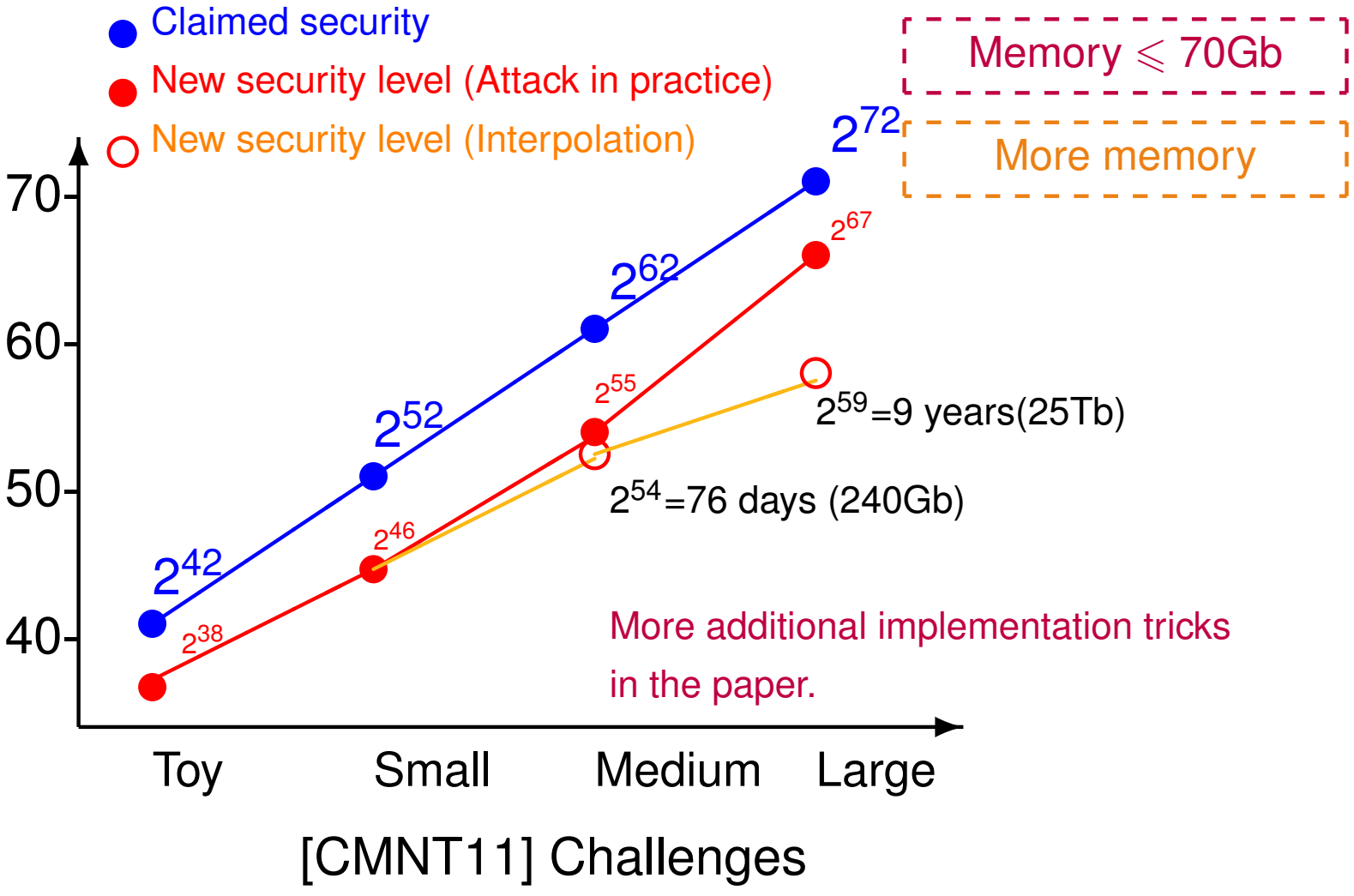
Result


Claimed security
70
60
50
40
2^{72}
2^{62}
2^{52}
2^{42}
Toy
Small
Medium
Large
[CMNT11] Challenges

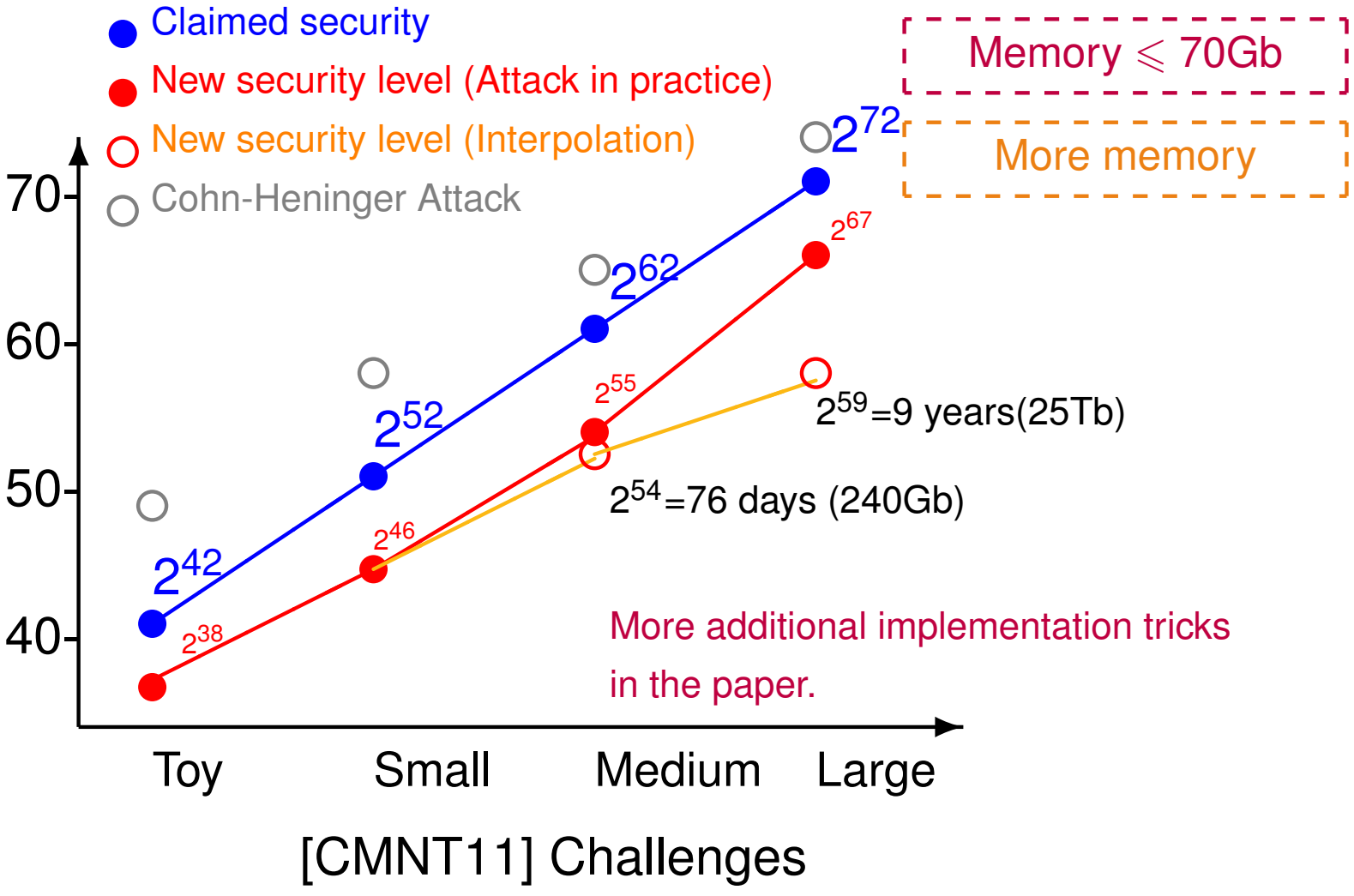
Result


Claimed security
New security level (Attack in practice)
Memory ⩽ 70Gb
70
60
50
40
2^{72}
2^{67}
2^{62}
2^{55}
2^{52}
2^{46}
2^{42}
2^{38}
More additional implementation tricks
in the paper.
Toy
Small
Medium
Large
[CMNT11] Challenges

Result


Claimed security
New security level (Attack in practice)
New security level (Interpolation)
Memory ⩽ 70Gb
More memory
70
60
50
40
2^{72}
2^{67}
2^{62}
2^{55}
2^{52}
2^{46}
2^{42}
2^{38}
2^{59}=9 years(25Tb)
2^{54}=76 days (240Gb)
More additional implementation tricks
in the paper.
Toy
Small
Medium
Large
[CMNT11] Challenges

Result


Claimed security
New security level (Attack in practice)
New security level (Interpolation)
Cohn-Heninger Attack
Memory ⩽ 70Gb
More memory
70
60
50
40
2^{72}
2^{67}
2^{62}
2^{55}
2^{52}
2^{46}
2^{42}
2^{38}
2^{59}=9 years(25Tb)
2^{54}=76 days (240Gb)
More additional implementation tricks
in the paper.
Toy
Small
Medium
Large
[CMNT11] Challenges

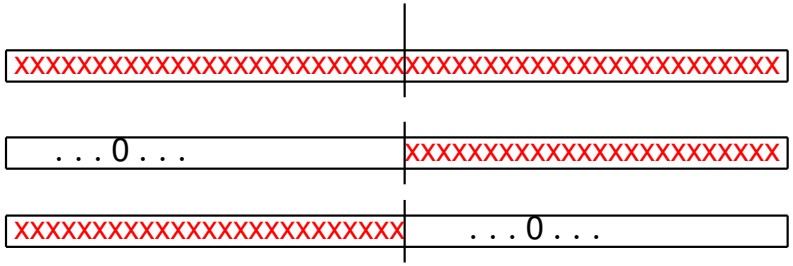
More than Approx-GCD

In general, if S is the direct sum of equal sized sets U and V,

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$\prod_S f(s_i)$

time/memory trade-off

$F(X) = \prod_i f(X + u_i) \longrightarrow \prod_j F(v_j).$

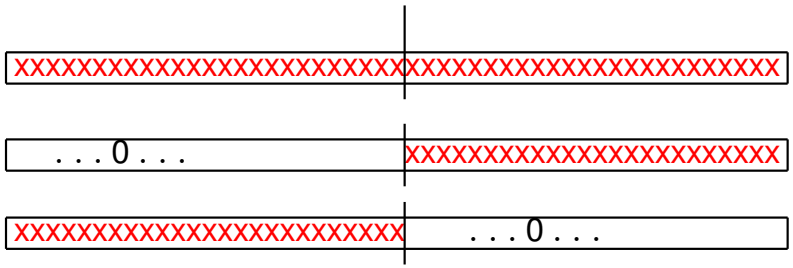
More than Approx-GCD

In general, if S is the direct sum of equal sized sets U and V,

$S = \{s_{i,j}\} = \{u_i + v_j\}$

$U = \{u_i\}$

$V = \{v_j\}$



$\prod_S f(s_i)$

time/memory trade-off

$F(X) = \prod_i f(X + u_i) \longrightarrow \prod_j F(v_j).$

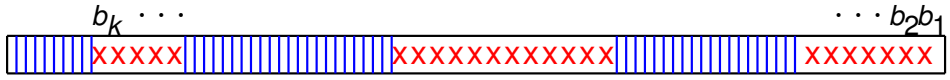
In particular, S doesn't have to be just consecutive numbers.

Applications in factoring, low-exponent RSA...

Noisy Factoring Problem: Key Recovery Attack

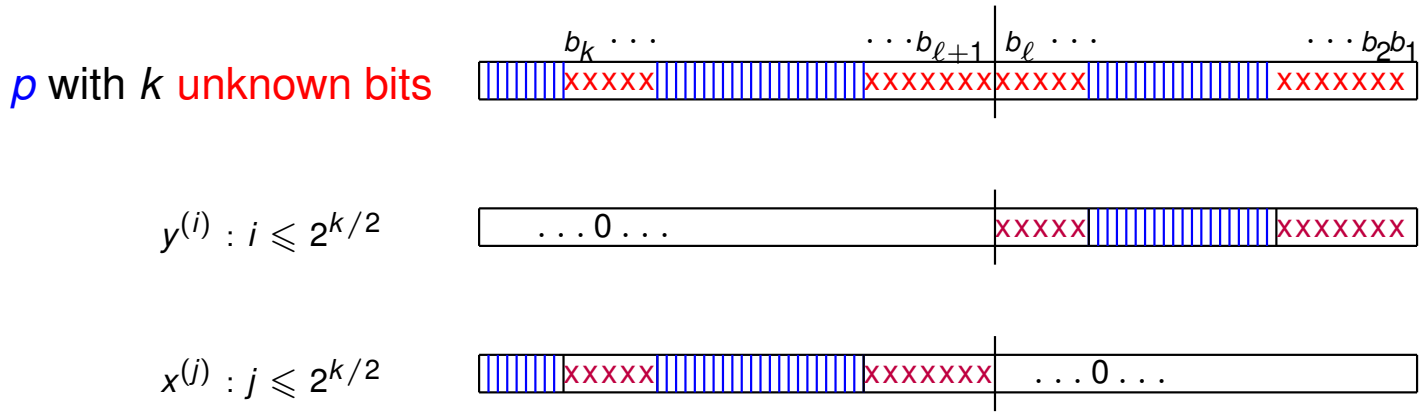
Consider $N = pq$, p is known except k bits. e.g. by side-channel attack.

p with k unknown bits



Noisy Factoring Problem: Key Recovery Attack

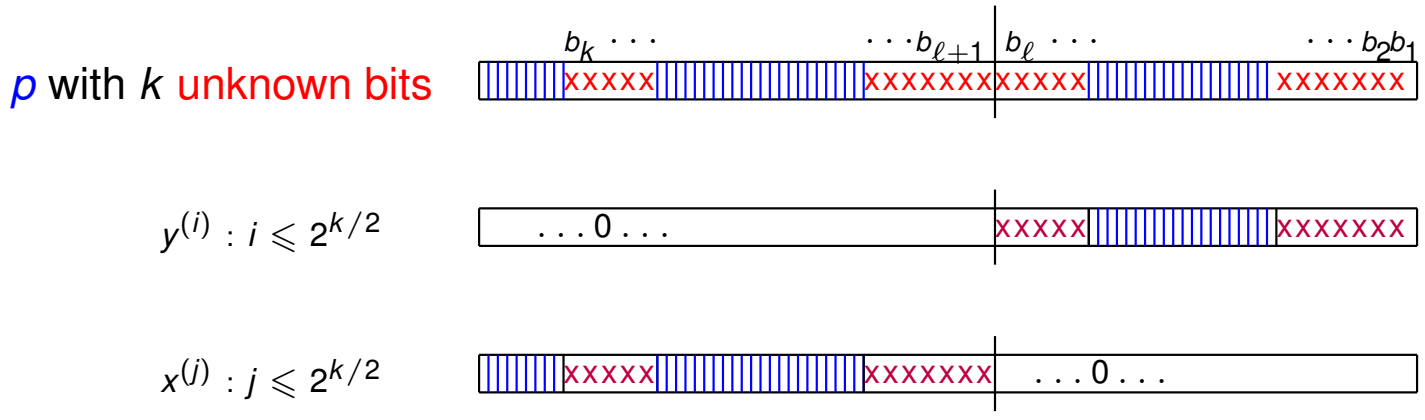
Consider $N = pq$, p is known except k bits. e.g. by side-channel attack.



Evaluate $f(X) = \prod_i (X + y^{(i)})$ on all $x^{(j)}$. Complexity: $\tilde{O}(2^{k/2})$

Noisy Factoring Problem: Key Recovery Attack

Consider $N = pq$, p is known except k bits. e.g. by side-channel attack.

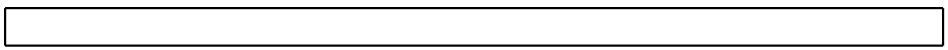


Evaluate $f(X) = \prod_i (X + y^{(i)})$ on all $x^{(j)}$. Complexity: $\tilde{O}(2^{k/2})$

Compared to Coppersmith's method, unknown bits do not have to be consecutive.

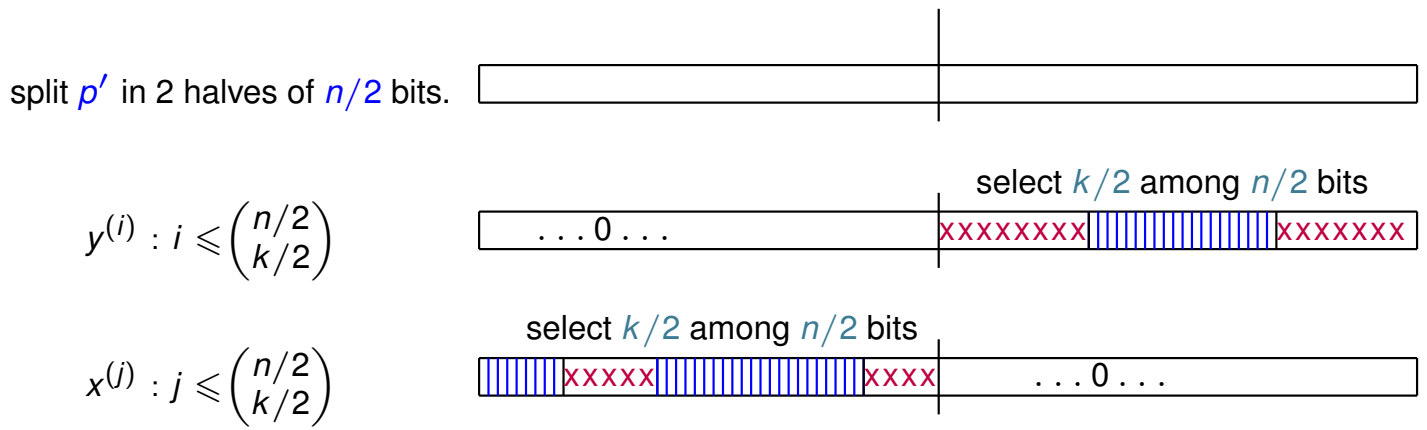
Noisy Factoring On Unknown Positions: Key Recovery Attack

Consider $N = pq$, we know p' which differs from p by k bits, whose positions are unknown.



Noisy Factoring On Unknown Positions: Key Recovery Attack

Consider $N = pq$, we know p' which differs from p by k bits, whose positions are unknown.



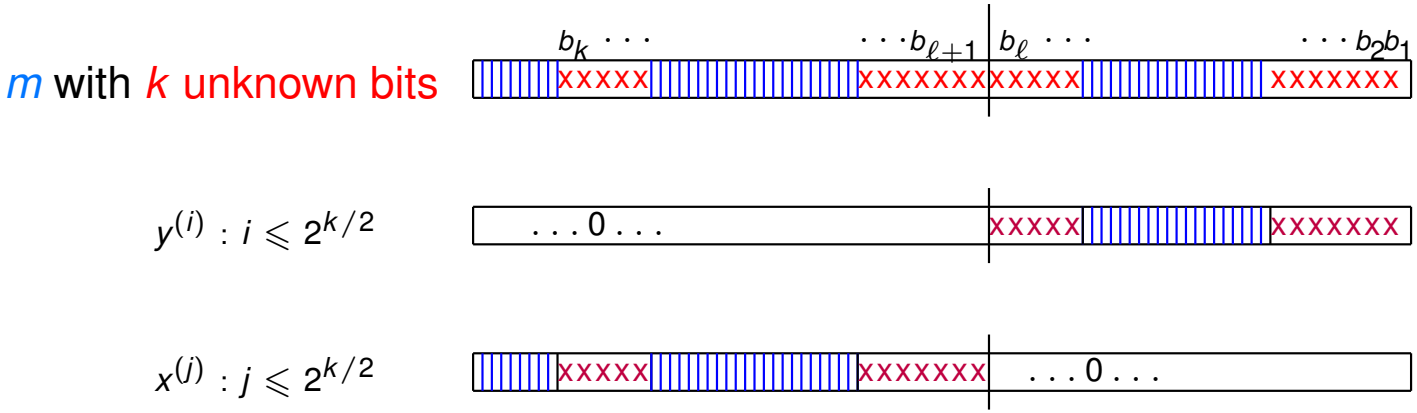
Evaluate $f(X) = \prod_i (X + y^{(i)})$ on all $x^{(j)}$.

Complexity: $\tilde{O}\left(\sqrt{k \cdot \binom{n}{k}}\right)$

RSA-CRT With Noisy Message: Key Recovery Attack

RSA-CRT signature: given a message m, and s its faulty RSA signature.
Key recovery attack: $p \leftarrow \gcd(s^e - m,\ N)$.

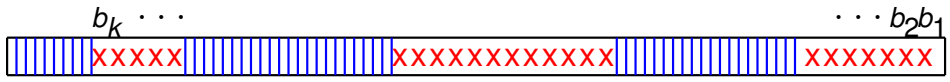
Now, our time/memory trade-off applies when m has k noisy bits (e.g. signature padding).



Low-Exponent RSA problem: Message Recovery Attack

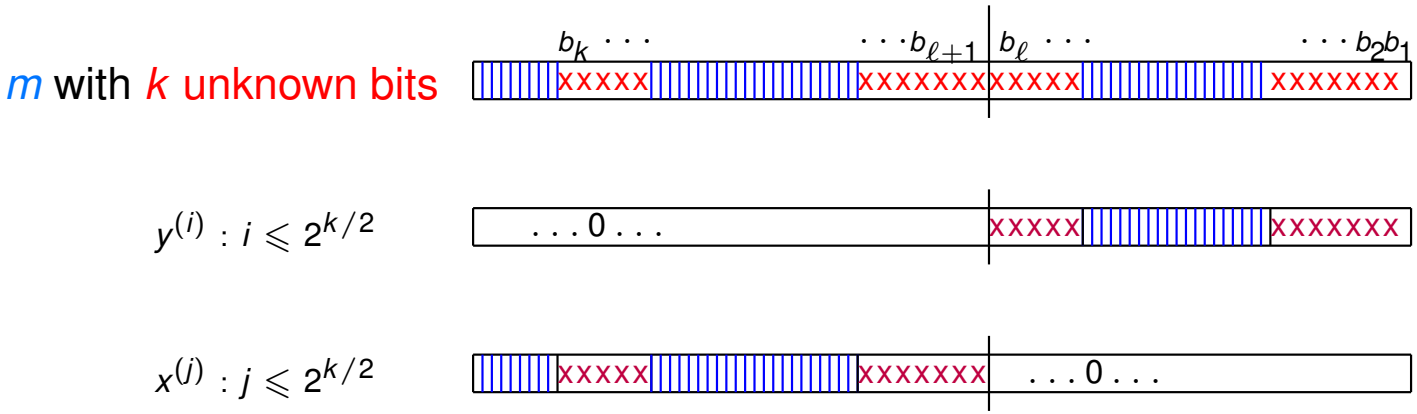
$c = m^e$ with e small.
After a decryption or fault attack, m is known except k bits, whose positions maybe unknown.

m with k unknown bits



Low-Exponent RSA problem: Message Recovery Attack

$c = m^e$ with e small.
After a decryption or fault attack, m is known except k bits, whose positions maybe unknown.



Evaluate $f(X) = \prod_{i=1}^{2^{k/2}} ((X + y^{(i)})^e - c)$ on all $x^{(j)}$.
Complexity: $\tilde{O}(2^{\frac{k+e}{2}})$

Conclusion

We can "square-root" the running time for "algebraic" exhaustive search.

1. Faster attacks on [vDGHV10] and [CMNT11] FHE schemes.
2. Other applications to cryptanalysis: noisy factoring and low-exponent RSA.

Conclusion

We can "square-root" the running time for "algebraic" exhaustive search.

1. Faster attacks on [vDGHV10] and [CMNT11] FHE schemes.
2. Other applications to cryptanalysis: noisy factoring and low-exponent RSA.

1. Is $\tilde{O}(2^{s/2})$ the best running time for $\prod_{i=0}^{2^s}(n+i)$?

Conclusion

We can "square-root" the running time for "algebraic" exhaustive search.

1. Faster attacks on [vDGHV10] and [CMNT11] FHE schemes.
2. Other applications to cryptanalysis: noisy factoring and low-exponent RSA.

1. Is $\tilde{O}(2^{s/2})$ the best running time for $\prod_{i=0}^{2^s}(n+i)$?
2. Is there a more efficient attack on Approx-GCD?

Conclusion

We can "square-root" the running time for "algebraic" exhaustive search.

1. Faster attacks on [vDGHV10] and [CMNT11] FHE schemes.
2. Other applications to cryptanalysis: noisy factoring and low-exponent RSA.

1. Is $\tilde{O}(2^{s/2})$ the best running time for $\prod_{i=0}^{2^s}(n+i)$?
2. Is there a more efficient attack on Approx-GCD?
3. Can we use more x_i to speed up the attack?

$$\begin{aligned} x_0 &= pq_0 \\ x_1 &= pq_1 + r_1 \\ x_2 &= pq_2 + r_2 \\ &\dots \end{aligned}$$

Analysis Of FHE Schemes

There are a lot of works on FHE schemes but very few on attacks.

If one wants to know how practical and secure is FHE, we need more work on attacks and concrete parameters.

Thank you!

Bibliography I

Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi.
Fully homomorphic encryption over the integers with shorter public keys.
In CRYPTO, pages 487–504, 2011.

Craig Gentry.
Fully homomorphic encryption using ideal lattices.
In STOC, pages 169–178, 2009.

Craig Gentry and Shai Halevi.
Implementing gentry's fully-homomorphic encryption scheme.
Cryptology ePrint Archive, Report 2010/520, 2010.
http://eprint.iacr.org/.

Nick Howgrave-Graham.
Approximate integer common divisors.
In Joseph Silverman, editor, Cryptography and Lattices, volume 2146 of Lecture Notes in Computer Science, pages 51–66. Springer Berlin / Heidelberg, 2001.

Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan.
Fully homomorphic encryption over the integers.
In EUROCRYPT, pages 24–43, 2010.