High Order Side-Channel Security for Elliptic-Curve Implementations

Sonia Belaïd and Matthieu Rivain

CHES 2023, September 13, Prague

Roadmap

- Case study: SCA on Montgomery ladder & countermeasures
- Our solution for high-order side-channel security
- Formal model and security proof
- Application and performances

Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$

Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \boldsymbol{\mathcal{O}}$

$R_1 \leftarrow \boldsymbol{P}$

for $i = n - 1$ **downto** 0 **do**

    $b \leftarrow k_i$

    $R_{1-b} \leftarrow R_{1-b} + R_b$

    $R_b \leftarrow 2 \cdot R_b$

end for

return R_0

Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$

Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $\boxed{R_0 \leftarrow \boldsymbol{\mathcal{O}}}$
2. $\boxed{R_1 \leftarrow \boldsymbol{P}}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad \boxed{R_{1-b} \leftarrow R_{1-b} + R_b}$
6. $\quad \boxed{R_b \leftarrow 2 \cdot R_b}$
7. **end for**
8. **return** $\boxed{R_0}$

Algebraic variables (EC points)

Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

Algebraic variables (EC points)

Index variables (scalar bits)

Case study: Montgomery ladder

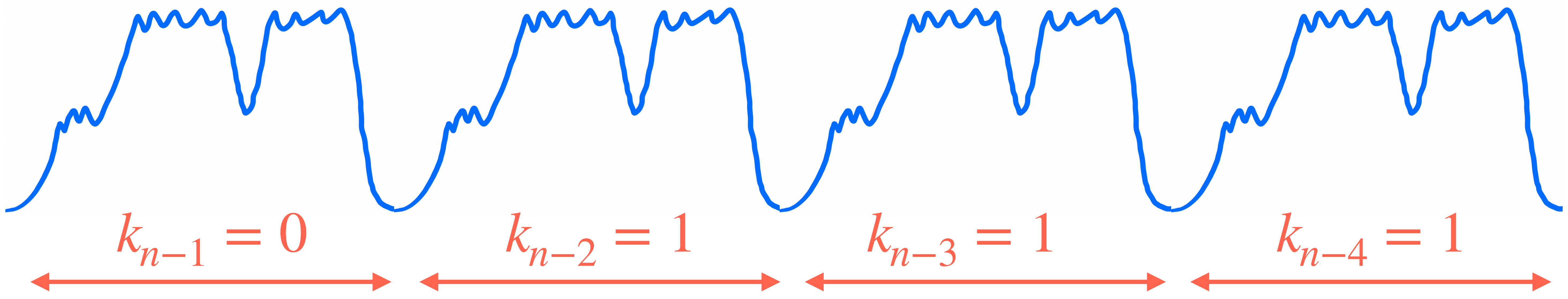
Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

Algebraic variables (EC points)

Index variables (scalar bits)



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

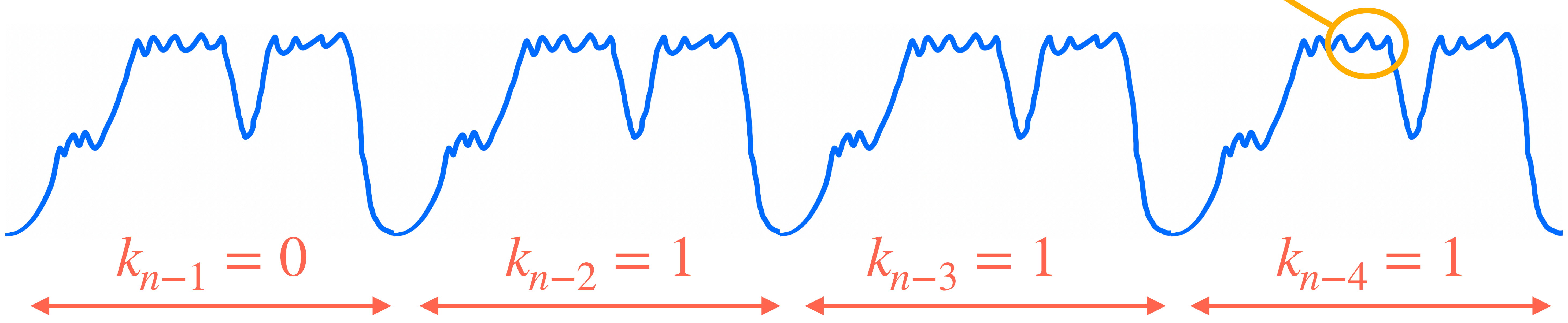
Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
  $b \leftarrow k_i$
  $R_{1-b} \leftarrow R_{1-b} + R_b$
  $R_b \leftarrow 2 \cdot R_b$
end for
return R_0

Algebraic variables (EC points)

Index variables (scalar bits)

Leakage on $R_0 = [7]P$



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$

Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

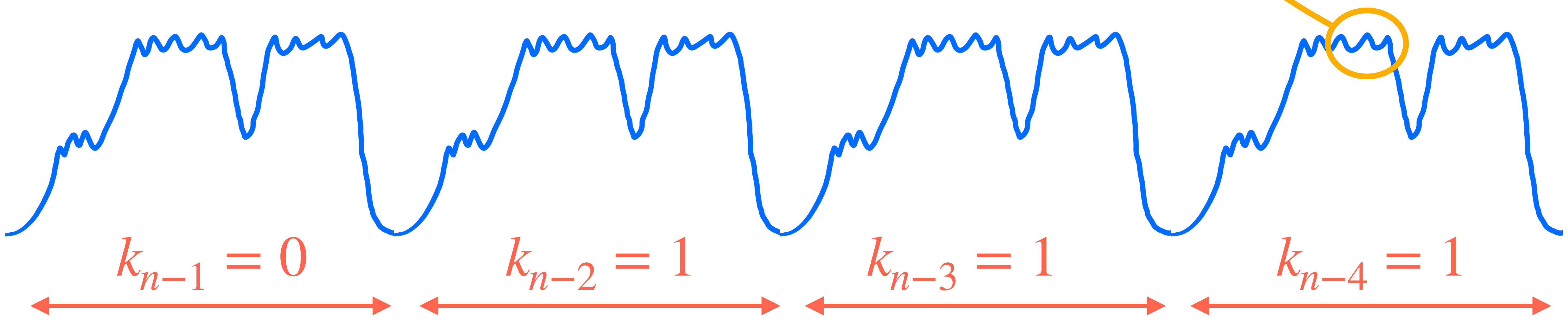
```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

Algebraic variables (EC points)

Index variables (scalar bits)

Compute correlation with $[0]P, \ldots, [15]P$

Leakage on $R_0 = [7]P$



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

Algebraic variables (EC points)

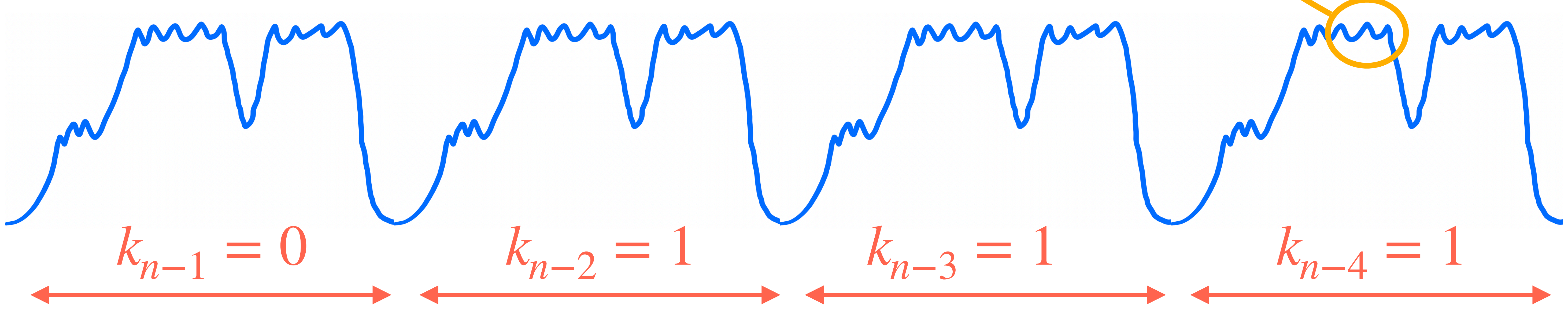
Index variables (scalar bits)

Leakage on $R_0 = [7]P$

Compute correlation with $[0]P, \ldots, [15]P$

$\Rightarrow$ Best correlation for $[7]P$

$\Rightarrow (k_{n-1}, \ldots, k_{n-3}) = (0,1,1,1)$



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n - 1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
6. $\quad R_b \leftarrow 2 \cdot R_b$
7. **end for**
8. **return** R_0

Algebraic variables (EC points)

Index variables (scalar bits)

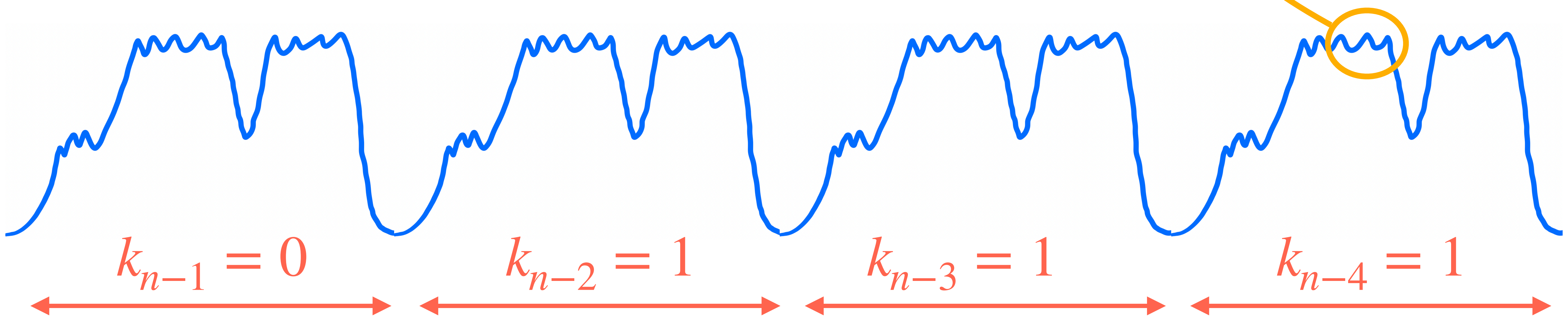
Leakage on $R_0 = [7]P$

Classic DPA Attack

Compute correlation with $[0]P, \ldots, [15]P$

$\Rightarrow$ Best correlation for $[7]P$

$\Rightarrow (k_{n-1}, \ldots, k_{n-3}) = (0,1,1,1)$



$k_{n-1} = 0$ $\quad k_{n-2} = 1$ $\quad k_{n-3} = 1$ $\quad k_{n-4} = 1$

Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

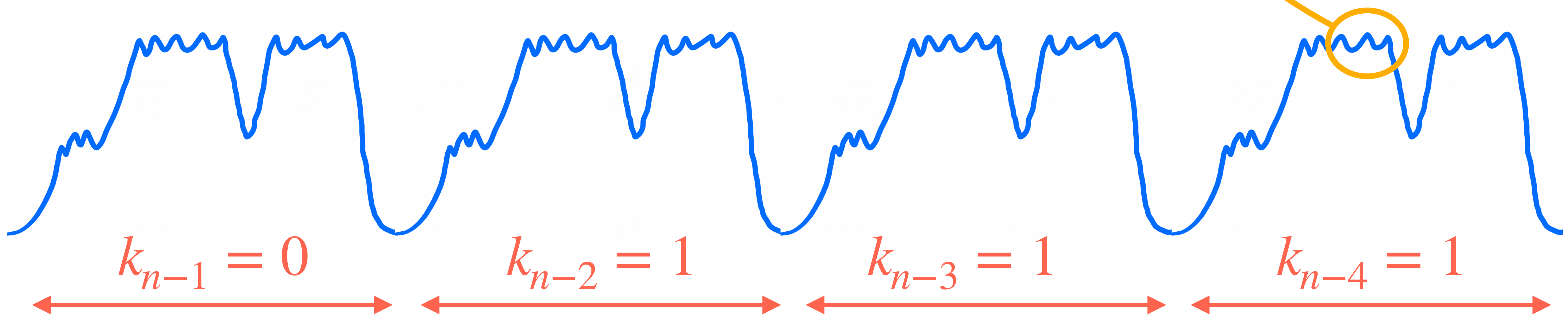
Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$\boxed{R_0 \leftarrow \mathcal{O}}$
$\boxed{R_1 \leftarrow \boldsymbol{P}}$
for $i = n - 1$ **downto** 0 **do**
  $\boxed{b \leftarrow k_i}$
  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return $\boxed{R_0}$

Algebraic variables (EC points)

Index variables (scalar bits)

Leakage on $R_0 = [7]P$



Classic DPA Attack

Compute correlation with $[0]P, \ldots, [15]P$

$\Rightarrow$ Best correlation for $[7]P$

$\Rightarrow (k_{n-1}, \ldots, k_{n-3}) = (0,1,1,1)$

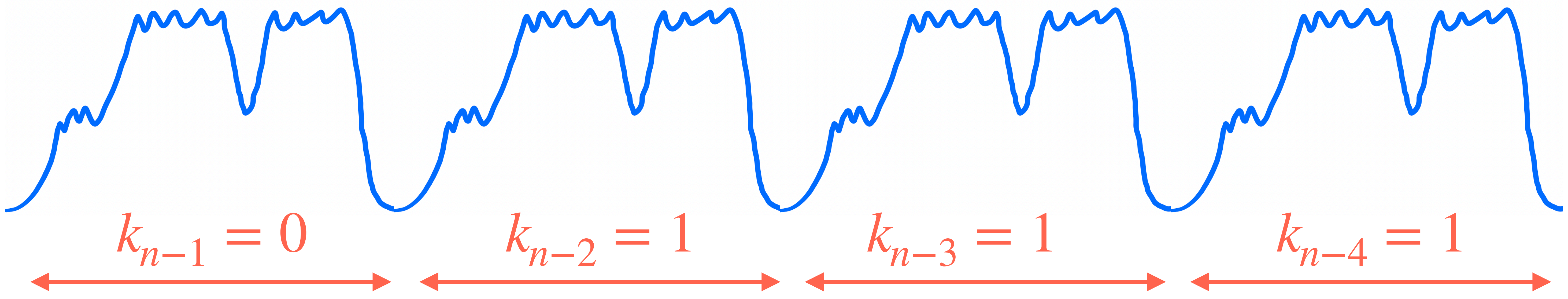
💡 Solution: Randomizing algebraic variables

Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

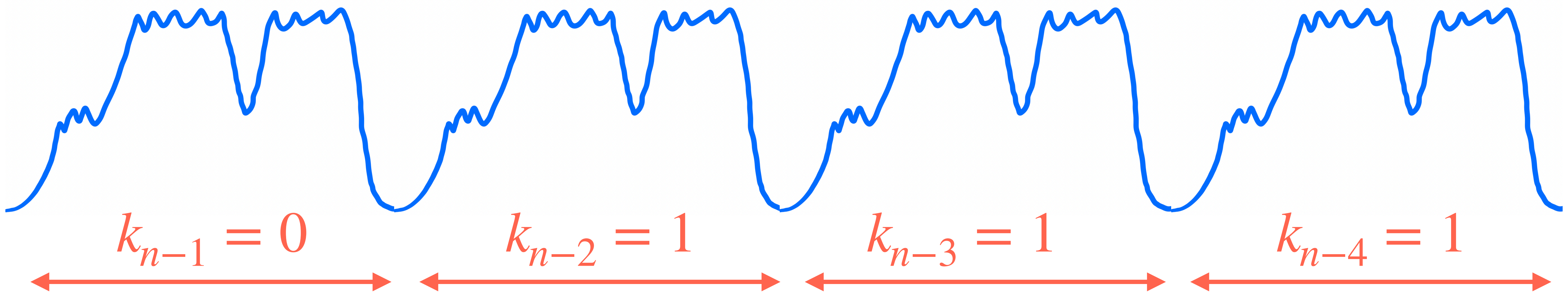


Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$\boxed{R_0 \leftarrow \mathcal{O}}$ Initial randomization
$\boxed{R_1 \leftarrow \boldsymbol{P}}$
for $i = n-1$ **downto** 0 **do**
  $b \leftarrow k_i$
  $R_{1-b} \leftarrow R_{1-b} + R_b$
  $R_b \leftarrow 2 \cdot R_b$
end for
return R_0



Case study: Montgomery ladder

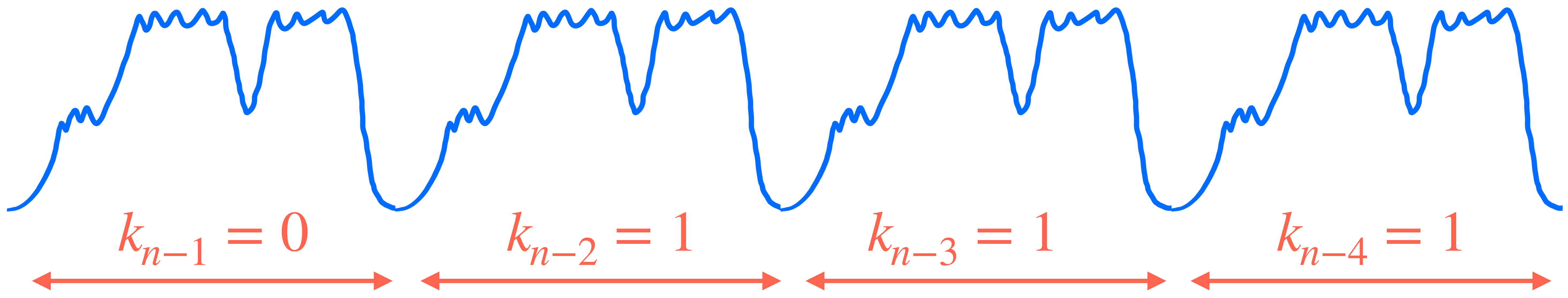
Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$\boxed{R_0 \leftarrow \mathcal{O}}$
$\boxed{R_1 \leftarrow \boldsymbol{P}}$
for $i = n-1$ **downto** 0 **do**
  $b \leftarrow k_i$
  $\boxed{R_{1-b} \leftarrow R_{1-b} + R_b}$
  $\boxed{R_b \leftarrow 2 \cdot R_b}$
end for
return $\boxed{R_0}$

Initial randomization

Propagation of
the randomization
(or re-randomization)



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

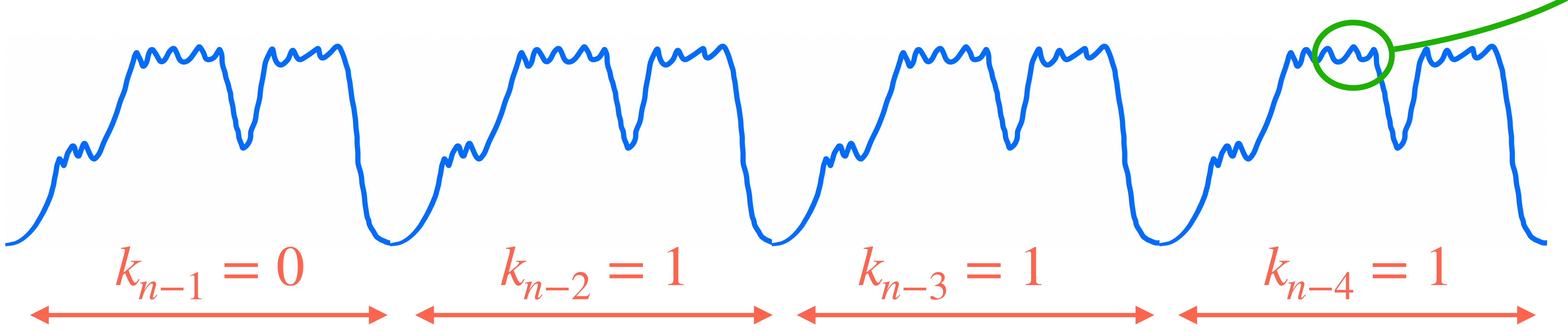
Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n-1$ **downto** 0 **do**
    $b \leftarrow k_i$
    $R_{1-b} \leftarrow R_{1-b} + R_b$
    $R_b \leftarrow 2 \cdot R_b$
end for
return R_0

Initial randomization

Propagation of
the randomization
(or re-randomization)

Leakage on
randomized R_0



Case study: Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

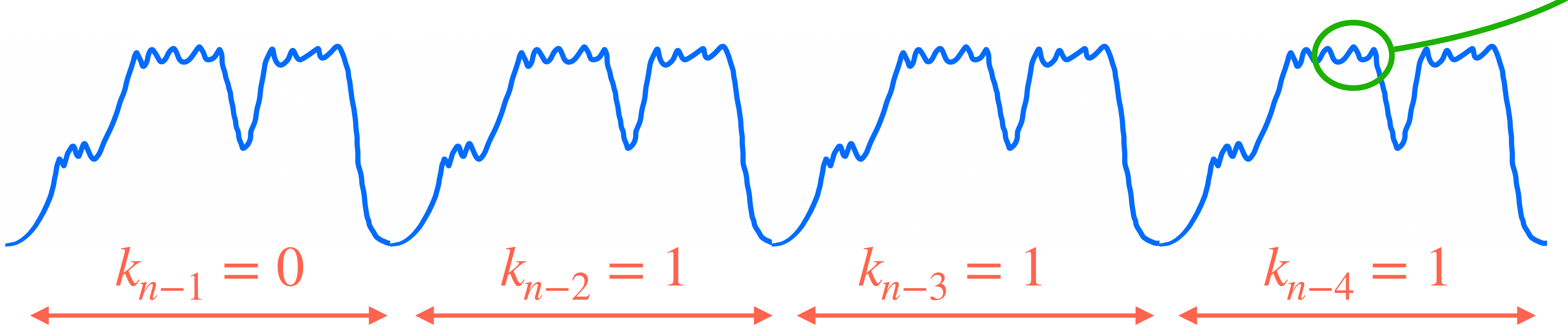
```
R_0 ← O
R_1 ← P
for i = n − 1 downto 0 do
    b ← k_i
    R_{1−b} ← R_{1−b} + R_b
    R_b ← 2 · R_b
end for
return R_0
```

Initial randomization

Propagation of the randomization (or re-randomization)

😇 No correlation anymore

Leakage on randomized R_0



Randomization techniques

- Randomization of the projective / Jacobian coordinates:
  - Point $P = (x, y)$ represented as $P \equiv (X : Y : Z)$ s.t. $x = X/Z$ and $y = Y/Z$
  - Random $r \leftarrow \mathbb{F}$, $\begin{cases} X' := r \cdot X \\ Y' := r \cdot Y \\ Z' := r \cdot Z \end{cases} \implies (X' : Y' : Z') \equiv P$

Randomization techniques

- Randomization of the projective / Jacobian coordinates:
  - Point $P = (x, y)$ represented as $P \equiv (X : Y : Z)$ s.t. $x = X/Z$ and $y = Y/Z$
  - Random $r \leftarrow \mathbb{F}$, $\begin{cases} X' := r \cdot X \\ Y' := r \cdot Y \\ Z' := r \cdot Z \end{cases} \implies (X' : Y' : Z') \equiv P$
- Randomisation of coordinates (field elements):
  - Elements of $\mathbb{F}_p$ (integers mod p) are represented modulo hp for some h
  - Random $r \leftarrow [0,h)$, $x' := x + r \cdot p \pmod{hp} \implies x' \equiv x \pmod{p}$

Randomization techniques

- Randomization of the projective / Jacobian coordinates:
  - Point $P = (x, y)$ represented as $P \equiv (X : Y : Z)$ s.t. $x = X/Z$ and $y = Y/Z$
  - Random $r \leftarrow \mathbb{F}$, $\begin{cases} X' := r \cdot X \\ Y' := r \cdot Y \\ Z' := r \cdot Z \end{cases} \implies (X' : Y' : Z') \equiv P$
- Randomisation of coordinates (field elements):
  - Elements of $\mathbb{F}_p$ (integers mod p) are represented modulo hp for some h
  - Random $r \leftarrow [0,h)$, $x' := x + r \cdot p \pmod{hp} \implies x' \equiv x \pmod{p}$

💡 Intuition: hard to break with common SC leakage

Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$

Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$

$R_1 \leftarrow \boldsymbol{P}$

for $i = n-1$ **downto** 0 **do**

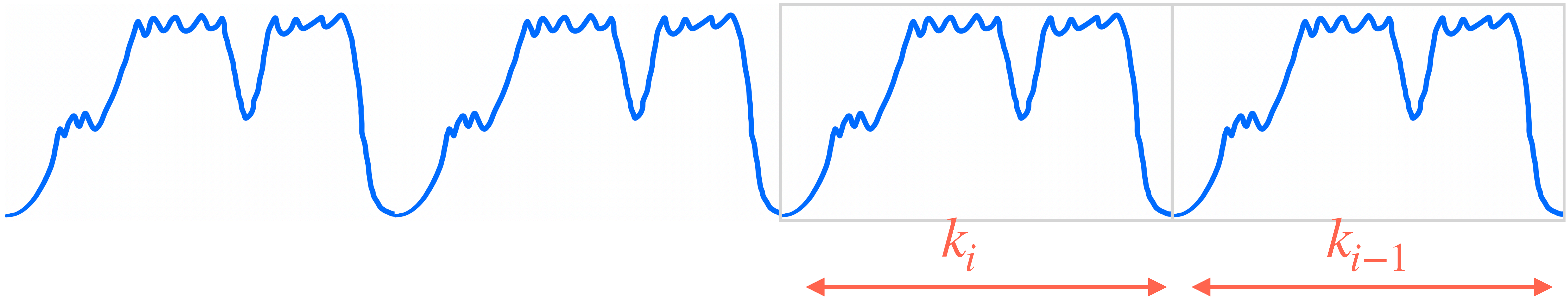
  $\boxed{b \leftarrow k_i}$

  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$

  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$

end for

return R_0

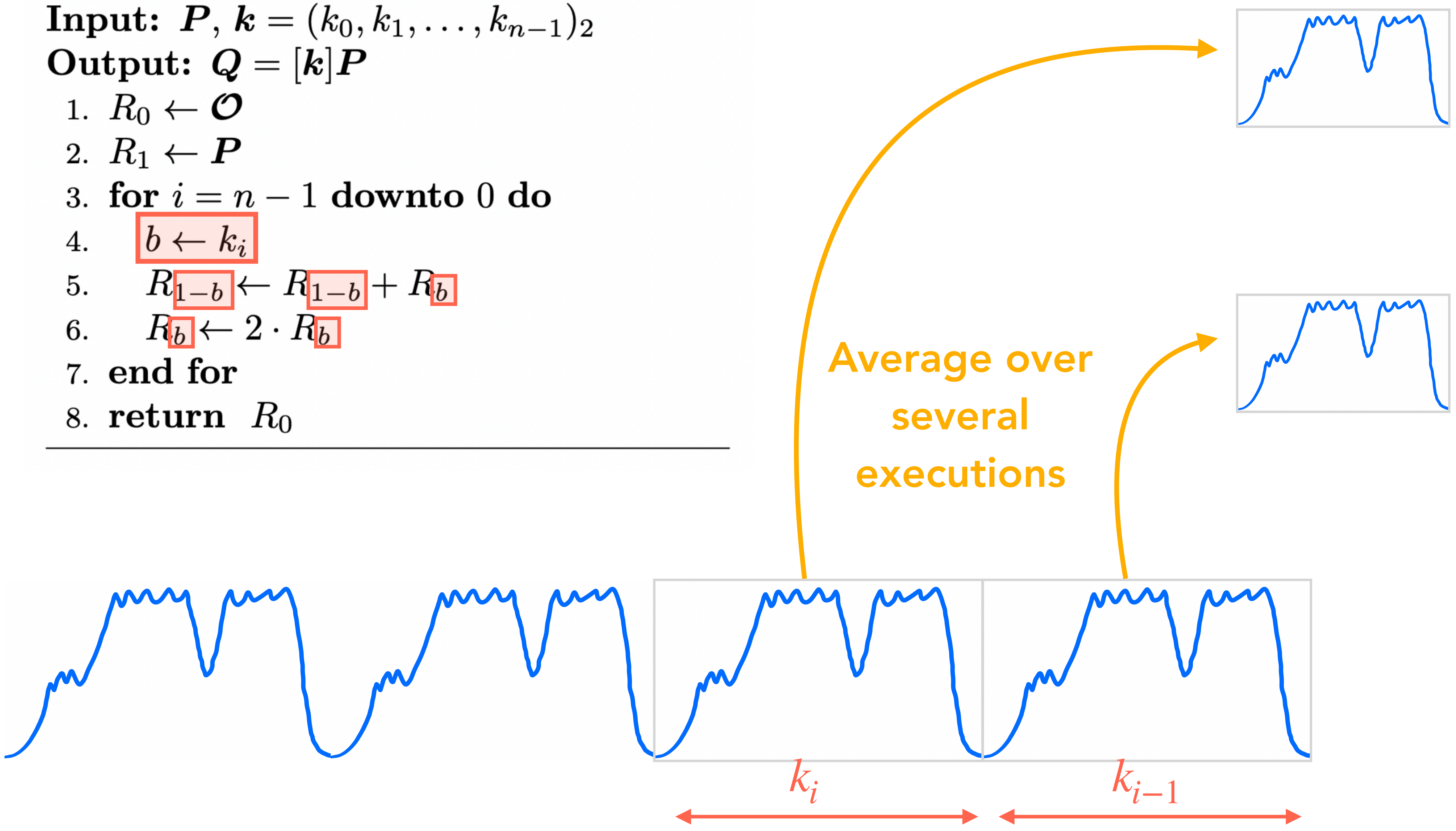


Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \boldsymbol{\mathcal{O}}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
  $\boxed{b \leftarrow k_i}$
  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return R_0

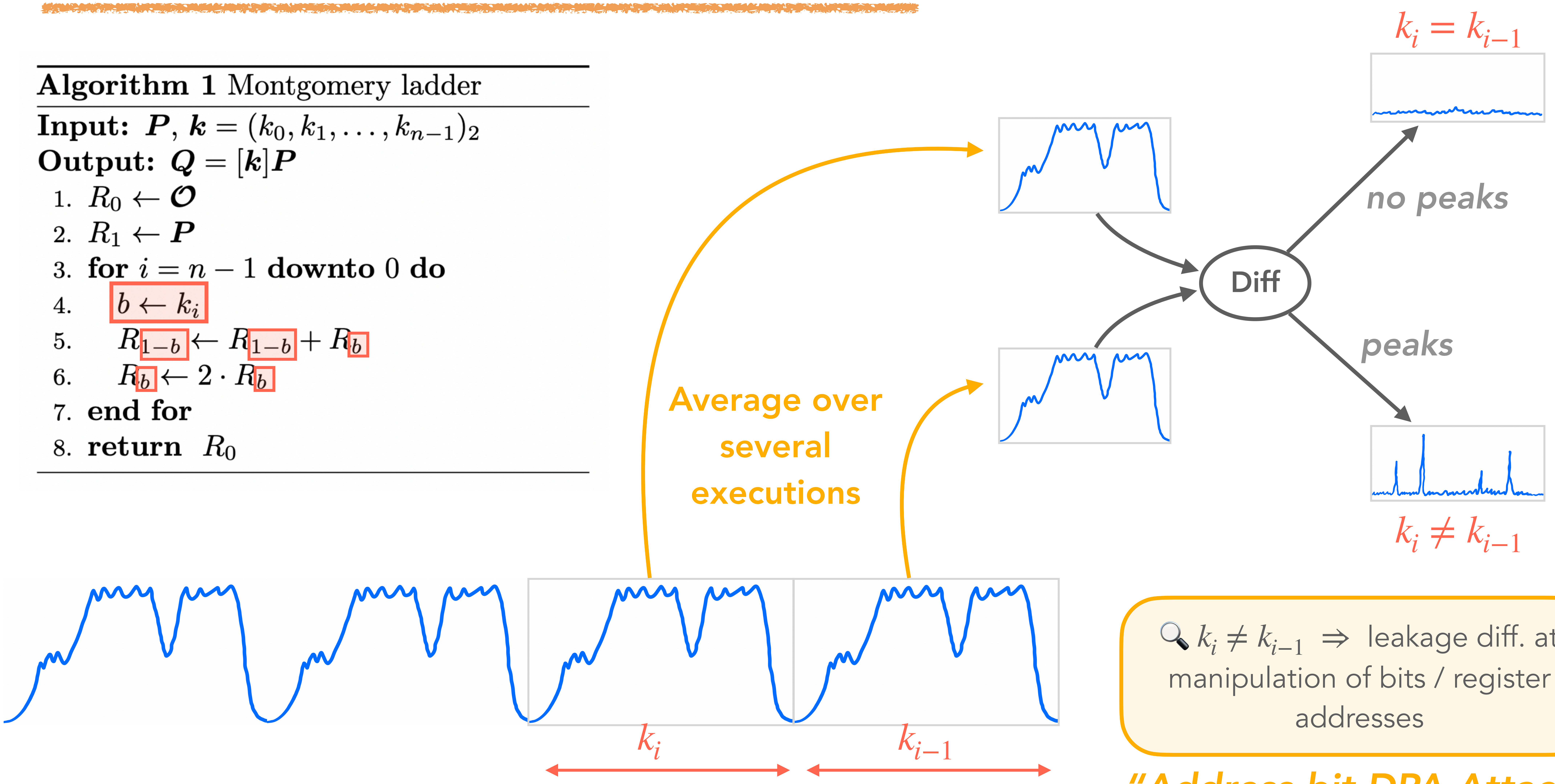


Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \boldsymbol{\mathcal{O}}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
  $\boxed{b \leftarrow k_i}$
  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return R_0



🔍 $k_i \neq k_{i-1} \Rightarrow$ leakage diff. at manipulation of bits / register addresses

"Address-bit DPA Attack"

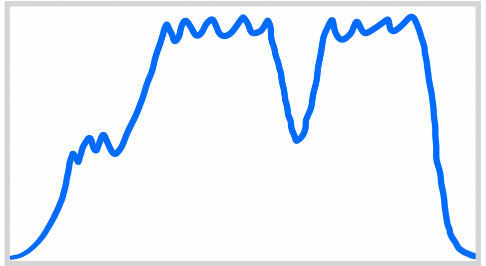
Back to Montgomery ladder

Algorithm 1 Montgomery ladder

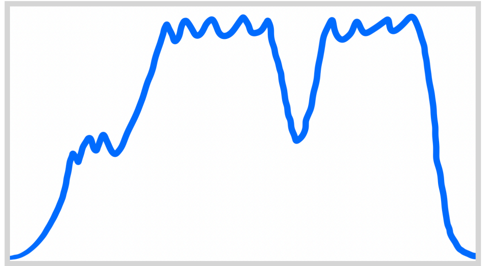
Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

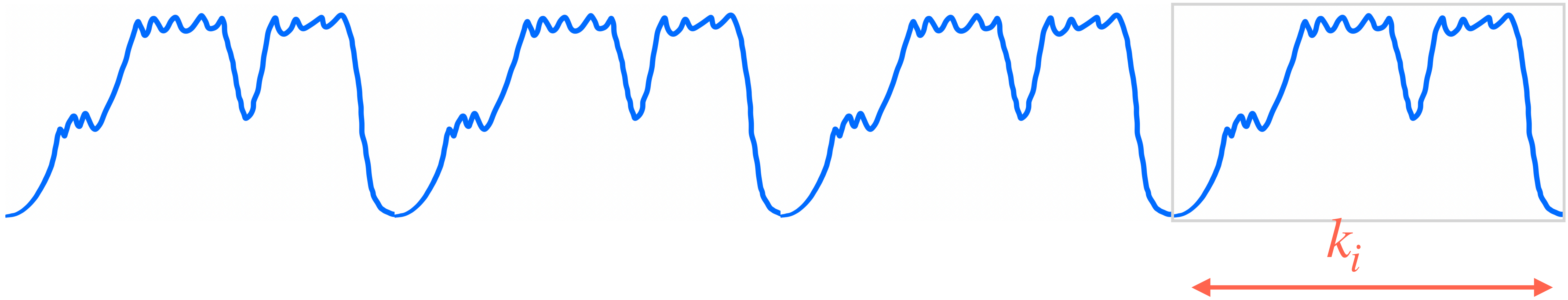
$R_0 \leftarrow \boldsymbol{\mathcal{O}}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n-1$ **downto** 0 **do**
  $\boxed{b \leftarrow k_i}$
  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return R_0

Precomputed template for $k_i = 0$



Precomputed template for $k_i = 1$



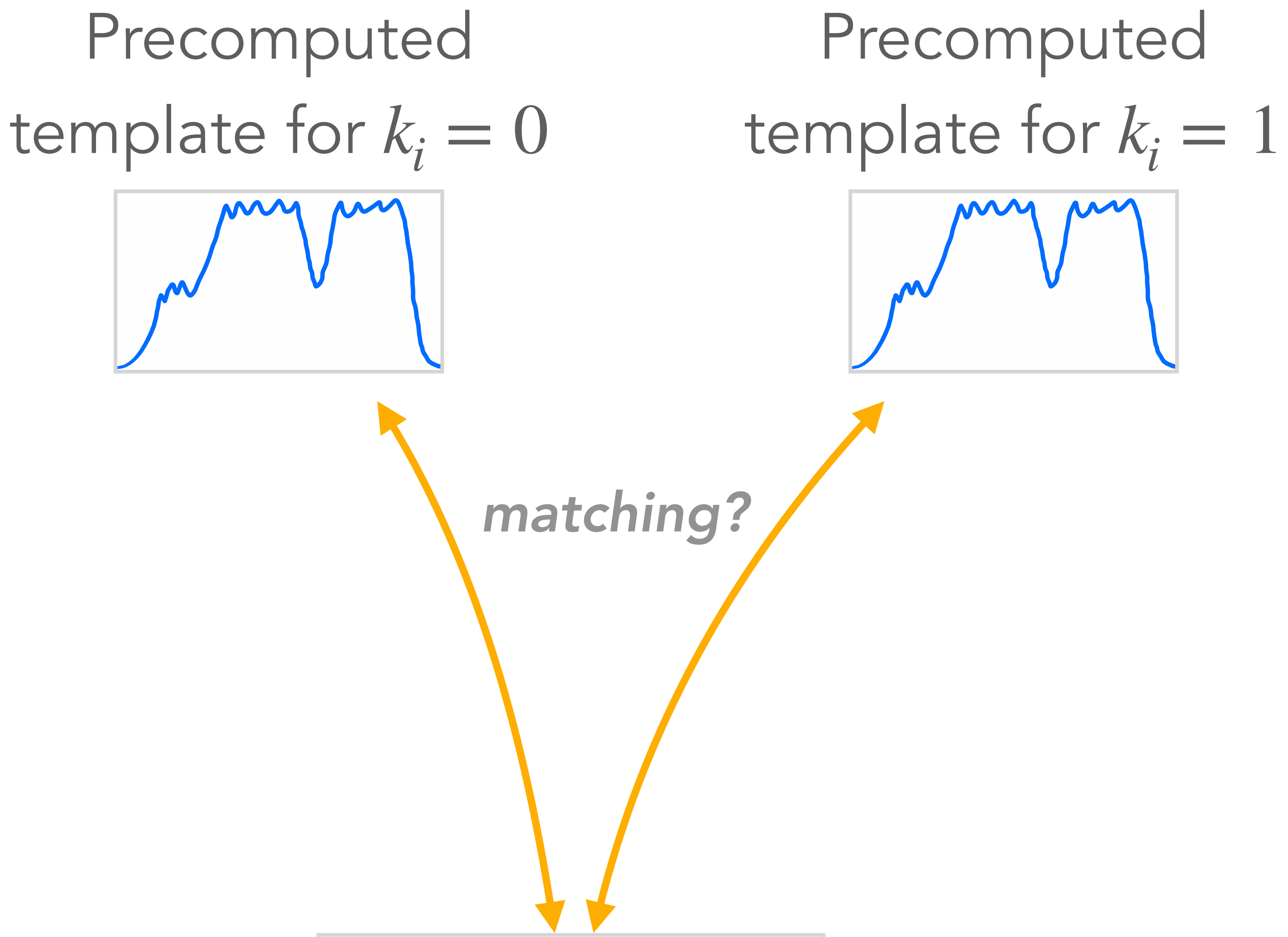


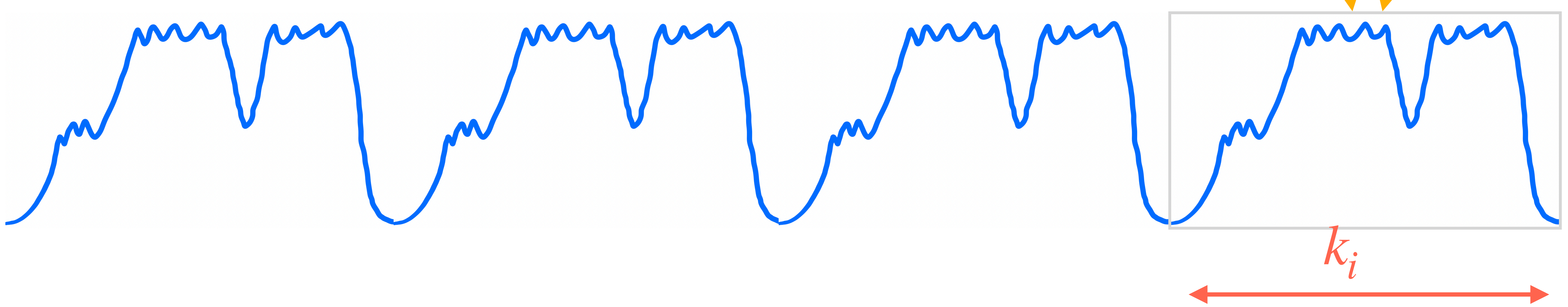
Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \boldsymbol{\mathcal{O}}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
  $\boxed{b \leftarrow k_i}$
  $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
  $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return R_0





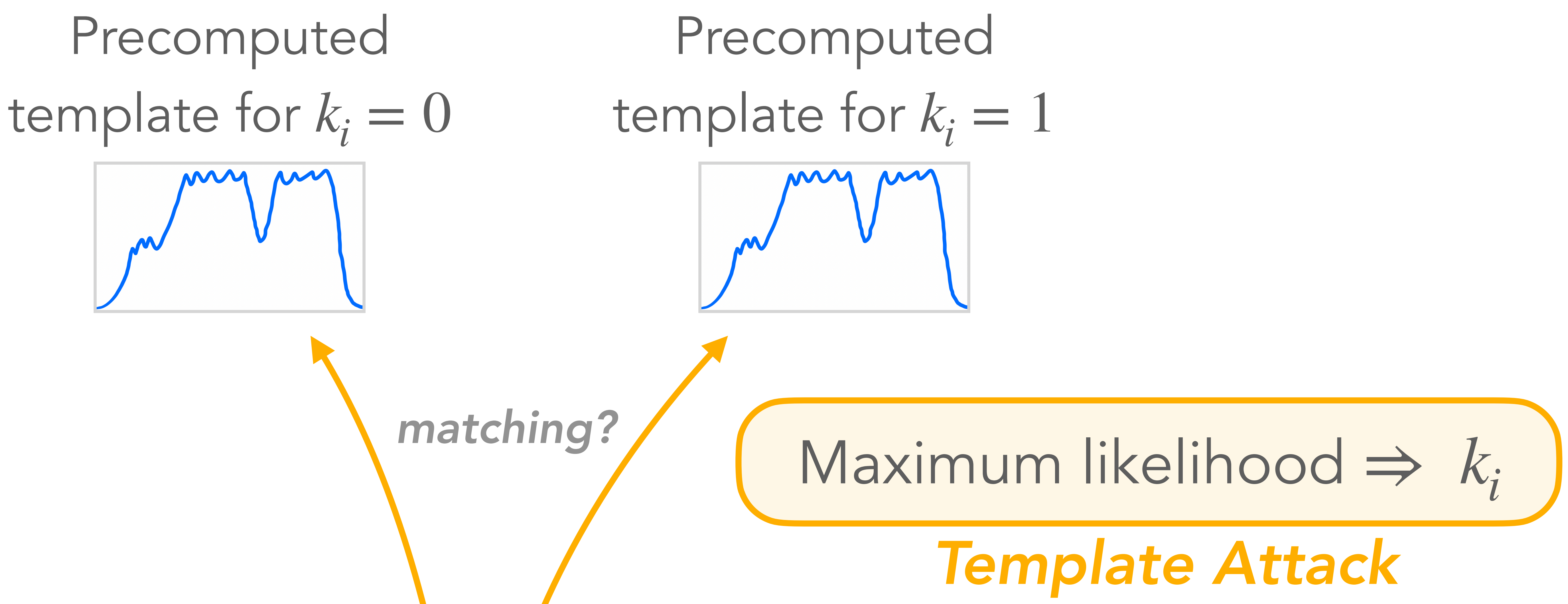
Back to Montgomery ladder

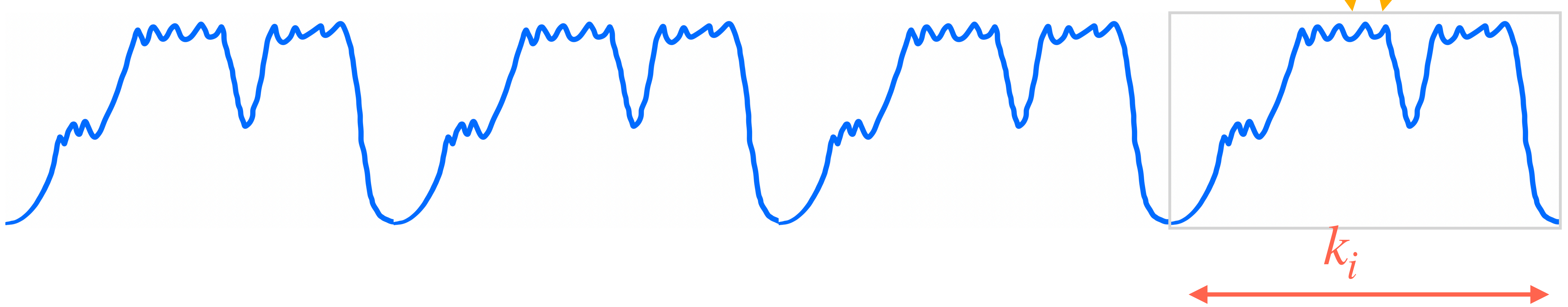
Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$

Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \boldsymbol{\mathcal{O}}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n - 1$ **downto** 0 **do**
4. $\quad \boxed{b \leftarrow k_i}$
5. $\quad R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
6. $\quad R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
7. **end for**
8. **return** R_0



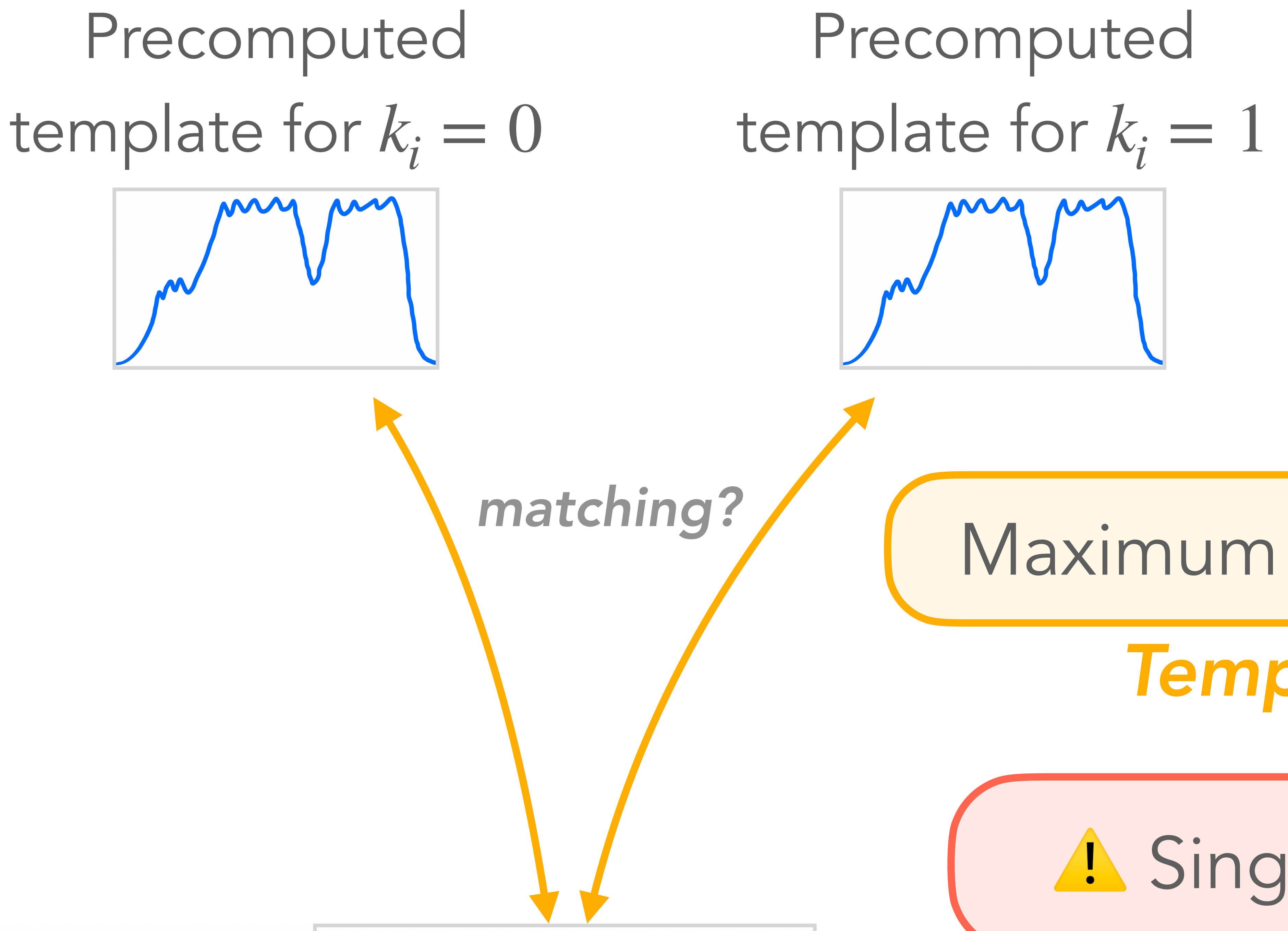


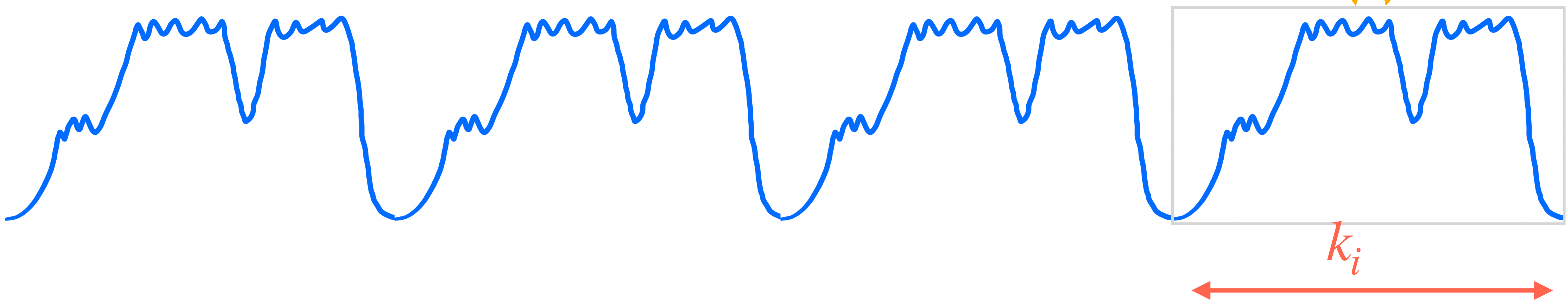
Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad \boxed{b \leftarrow k_i}$
5. $\quad R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
6. $\quad R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
7. **end for**
8. **return** R_0





Maximum likelihood $\Rightarrow$ k_i

Template Attack

⚠️ Single trace attack

Back to Montgomery ladder

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \mathcal{O}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
    $\boxed{b \leftarrow k_i}$
    $R_{\boxed{1-b}} \leftarrow R_{\boxed{1-b}} + R_{\boxed{b}}$
    $R_{\boxed{b}} \leftarrow 2 \cdot R_{\boxed{b}}$
end for
return R_0

Maximum likelihood $\Rightarrow$ k_i

Template Attack

⚠️ Single trace attack

💡 Solution: Randomizing the scalar

Scalar randomization

- Scalar blinding:

$$k' \leftarrow k + r \cdot |E(\mathbb{F}_p)| \quad \Longrightarrow \quad [k]P = [k']P$$

with $|E(\mathbb{F}_p)|$ the order of the EC

Scalar randomization

- Scalar blinding:

$$k' \leftarrow k + r \cdot |E(\mathbb{F}_p)| \quad \Longrightarrow \quad [k]P = [k']P$$

with $|E(\mathbb{F}_p)|$ the order of the EC

- Scalar splitting:

$$\begin{cases} Q_1 = [k-r]P \\ Q_2 = [r]P \end{cases} \quad \Longrightarrow \quad [k]P = Q_1 + Q_2$$

Scalar randomization

- Scalar blinding:

$$k' \leftarrow k + r \cdot |E(\mathbb{F}_p)| \quad \Longrightarrow \quad [k]P = [k']P$$

with $|E(\mathbb{F}_p)|$ the order of the EC

- Scalar splitting:

$$\begin{cases} Q_1 = [k - r]P \\ Q_2 = [r]P \end{cases} \quad \Longrightarrow \quad [k]P = Q_1 + Q_2$$

⚠️ Still vulnerable to single trace attack

Scalar randomization

- <u>Boolean masking:</u>

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

$R_0 \leftarrow \boldsymbol{\mathcal{O}}$
$R_1 \leftarrow \boldsymbol{P}$
for $i = n - 1$ **downto** 0 **do**
    $b \leftarrow k_i$
    $R_{1-b} \leftarrow R_{1-b} + R_b$
    $R_b \leftarrow 2 \cdot R_b$
end for
return R_0

Scalar randomization

- Boolean masking:

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
6. $\quad R_b \leftarrow 2 \cdot R_b$
7. **end for**
8. **return** R_0

Relabel $\begin{cases} T_0 := R_b \\ T_1 := R_{1-b} \end{cases}$

$T_1 \leftarrow T_1 + T_0$
$T_0 \leftarrow 2 \cdot T_0$
If $(b = 1)$ **then** Swap(T_0, T_1)

Scalar randomization

- Boolean masking:

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
6. $\quad R_b \leftarrow 2 \cdot R_b$
7. **end for**
8. **return** R_0

Relabel $\begin{cases} T_0 := R_b \\ T_1 := R_{1-b} \end{cases}$

$T_1 \leftarrow T_1 + T_0$
$T_0 \leftarrow 2 \cdot T_0$
If $(b = 1)$ **then** Swap(T_0, T_1)

Conditional swap

CSwap(T_0, T_1, b):

1. $(S_0, S_1) \leftarrow (T_0, T_1)$
2. $T_0 = S_b$
3. $T_1 = S_{1-b}$

Scalar randomization

- Boolean masking:

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
6. $\quad R_b \leftarrow 2 \cdot R_b$
7. **end for**
8. **return** R_0

Relabel $\begin{cases} T_0 := R_b \\ T_1 := R_{1-b} \end{cases}$

$T_1 \leftarrow T_1 + T_0$
$T_0 \leftarrow 2 \cdot T_0$
If $(b = 1)$ **then** Swap(T_0, T_1)

Conditional swap

CSwap(T_0, T_1, b):

1. $(S_0, S_1) \leftarrow (T_0, T_1)$
2. $T_0 = S_b$
3. $T_1 = S_{1-b}$

☝ Only operation manipulating $b = k_i$

Scalar randomization

- Boolean masking:

Algorithm 1 Montgomery ladder

Input: $\boldsymbol{P}$, $\boldsymbol{k} = (k_0, k_1, \ldots, k_{n-1})_2$
Output: $\boldsymbol{Q} = [\boldsymbol{k}]\boldsymbol{P}$

1. $R_0 \leftarrow \mathcal{O}$
2. $R_1 \leftarrow \boldsymbol{P}$
3. **for** $i = n-1$ **downto** 0 **do**
4. $\quad b \leftarrow k_i$
5. $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
6. $\quad R_b \leftarrow 2 \cdot R_b$
7. **end for**
8. **return** R_0

Relabel $\begin{cases} T_0 := R_b \\ T_1 := R_{1-b} \end{cases}$

$T_1 \leftarrow T_1 + T_0$
$T_0 \leftarrow 2 \cdot T_0$
If $(b = 1)$ **then** Swap(T_0, T_1)

Conditional swap

CSwap(T_0, T_1, b):

1. $(S_0, S_1) \leftarrow (T_0, T_1)$
2. $T_0 = S_b$
3. $T_1 = S_{1-b}$

☝️ Only operation manipulating $b = k_i$

- Masking the scalar $(b^0, b^1) := (b \oplus r, r)$ for random bit $r \leftarrow \{0,1\}$
- Masked CSwap: $\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \end{cases} \iff \text{CSwap}(T_0, T_1, b)$

What can go wrong now?!

😼 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \text{ depends on } b$$

$$\Longrightarrow \text{2nd-order address-bit / template attack}$$

What can go wrong now?!

😺 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \text{ depends on } b$$
$$\implies \text{2nd-order address-bit / template attack}$$

🐭 2nd-order masking: $b = b^0 \oplus b^1 \oplus b^2$

$$\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \\ \text{CSwap}(T_0, T_1, b^2) \end{cases} \iff \text{CSwap}(T_0, T_1, b)$$

What can go wrong now?!

😺 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \ \text{ depends on } \ b$$
$$\implies \text{2nd-order address-bit / template attack}$$

🐭 2nd-order masking: $b = b^0 \oplus b^1 \oplus b^2$

$$\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \\ \text{CSwap}(T_0, T_1, b^2) \end{cases} \iff \quad \text{CSwap}(T_0, T_1, b)$$

😼 3rd-order attack $\Rightarrow$ 🐭 3rd-order masking $\Rightarrow \ldots \Rightarrow$ 😿 d-th order attack

What can go wrong now?!

😸 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \;\text{ depends on }\; b$$
$$\implies \text{2nd-order address-bit / template attack}$$

🐭 2nd-order masking: $b = b^0 \oplus b^1 \oplus b^2$

$$\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \\ \text{CSwap}(T_0, T_1, b^2) \end{cases} \iff \text{CSwap}(T_0, T_1, b)$$

exponentially hard in d

😼 3rd-order attack $\Rightarrow$ 🐭 3rd-order masking $\Rightarrow \ldots \Rightarrow$ 😿 d-th order attack

What can go wrong now?!

😼 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \text{ depends on } b$$

$$\Longrightarrow \text{ 2nd-order address-bit / template attack}$$

🐭 2nd-order masking: $b = b^0 \oplus b^1 \oplus b^2$

$$\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \\ \text{CSwap}(T_0, T_1, b^2) \end{cases} \iff \text{CSwap}(T_0, T_1, b)$$

exponentially hard in d

😼 3rd-order attack ⇒ 🐭 3rd-order masking ⇒ … ⇒ 😿 d-th order attack

😇 High order security
😇 Linear complexity in d
(only for swaps)

What can go wrong now?!

😼 2nd-order attack on masked scalar bits

$$\text{Leakage}(b^0) + \text{Leakage}(b^1) \ \text{ depends on } \ b$$
$$\implies \text{2nd-order address-bit / template attack}$$

🐭 2nd-order masking: $b = b^0 \oplus b^1 \oplus b^2$

$$\begin{cases} \text{CSwap}(T_0, T_1, b^0) \\ \text{CSwap}(T_0, T_1, b^1) \\ \text{CSwap}(T_0, T_1, b^2) \end{cases} \iff \text{CSwap}(T_0, T_1, b)$$

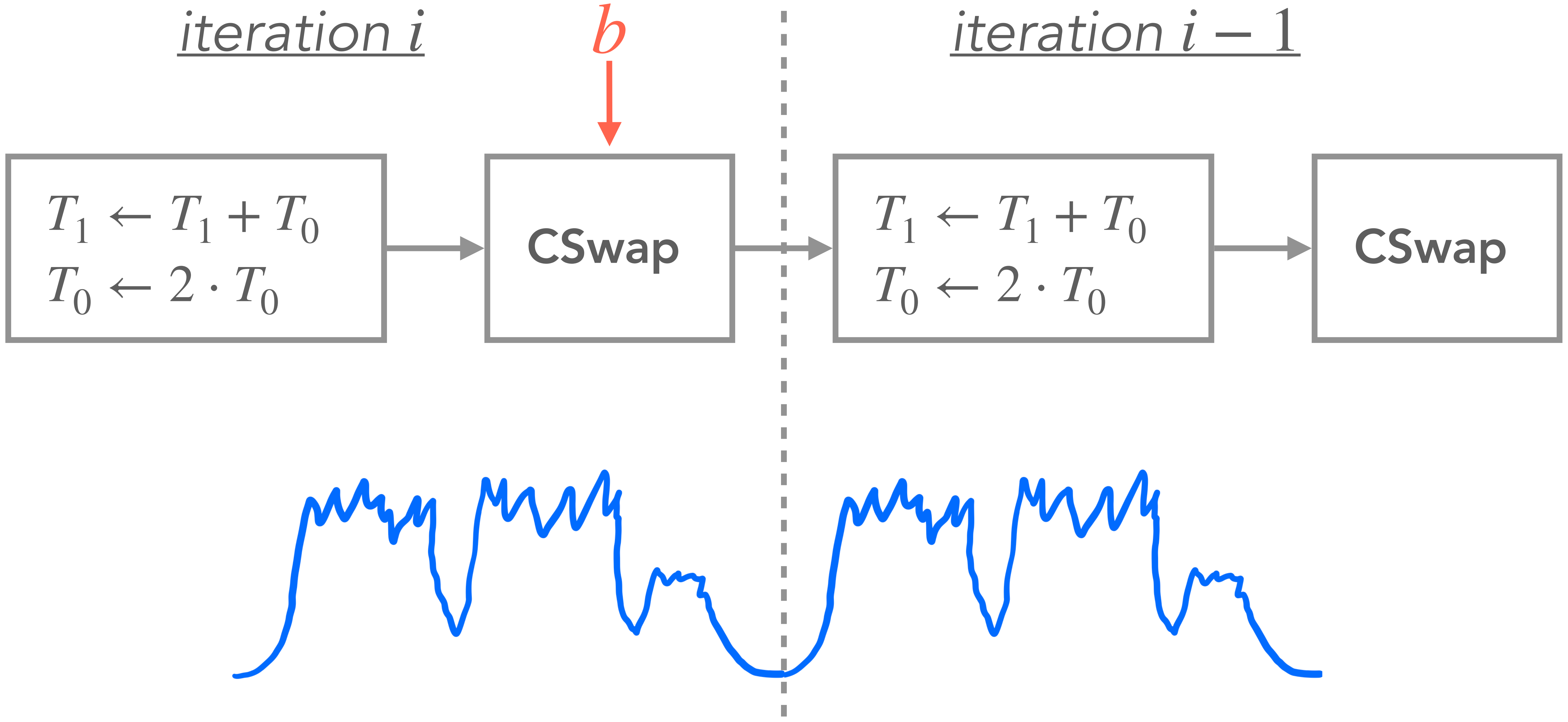
exponentially hard in d

😼 3rd-order attack $\Rightarrow$ 🐭 3rd-order masking $\Rightarrow \ldots \Rightarrow$ 😿 d-th order attack

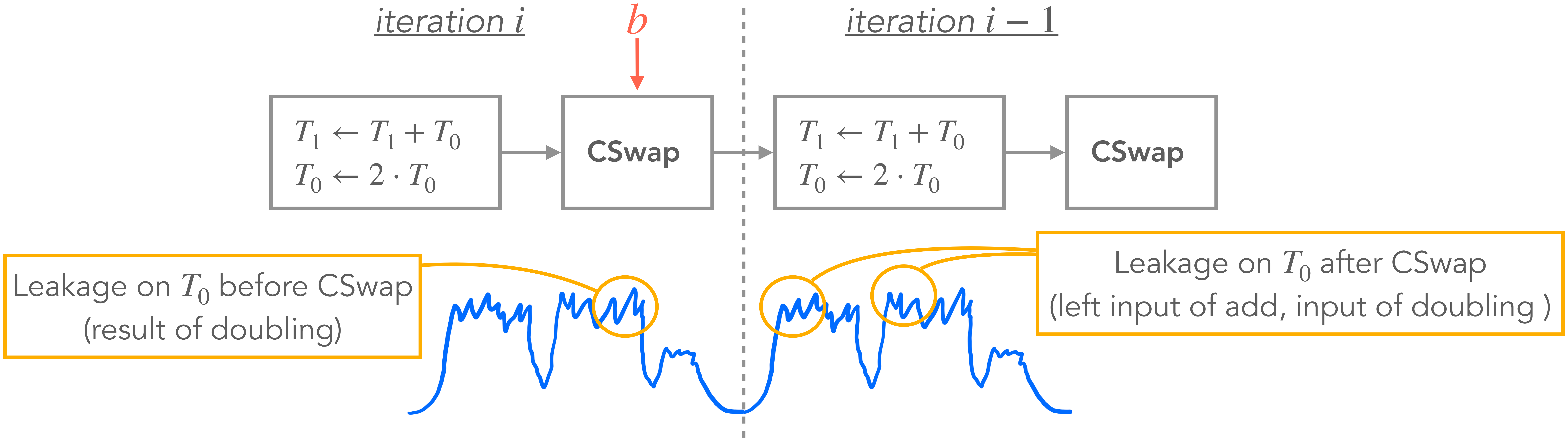
😈 But 2nd order "collision" leakage remains

😇 High order security
😇 Linear complexity in d
(only for swaps)

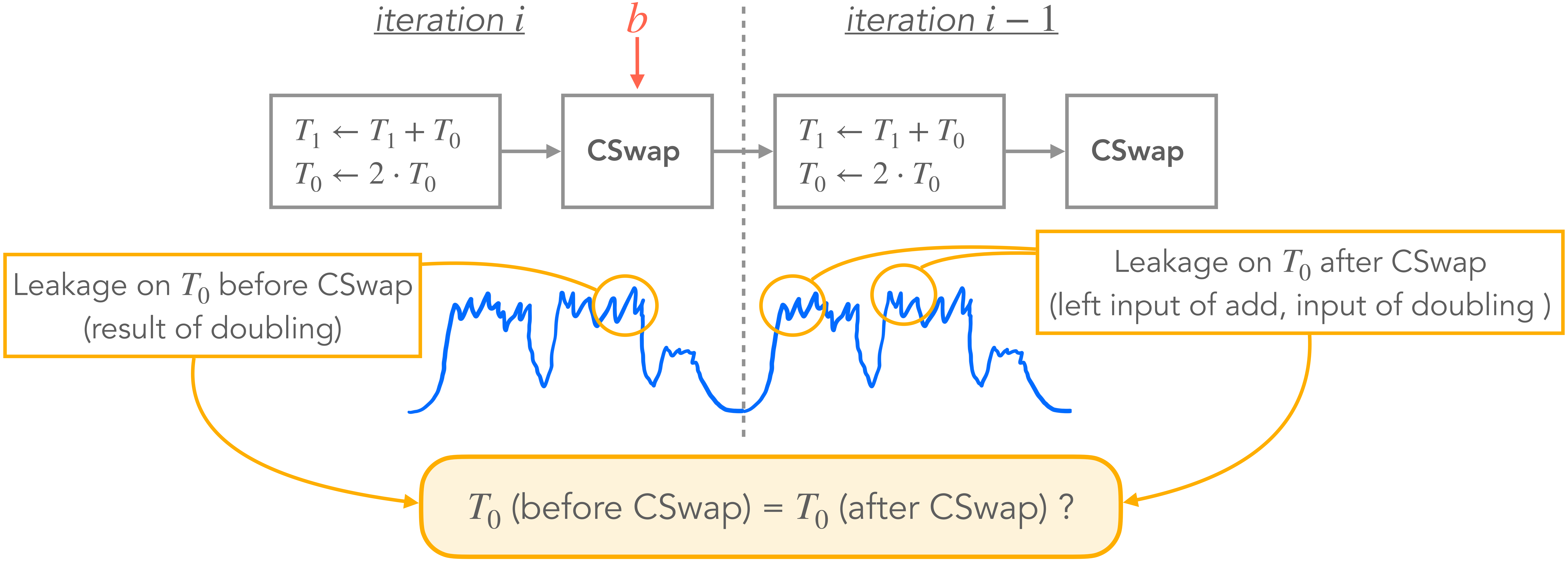
Collision attacks



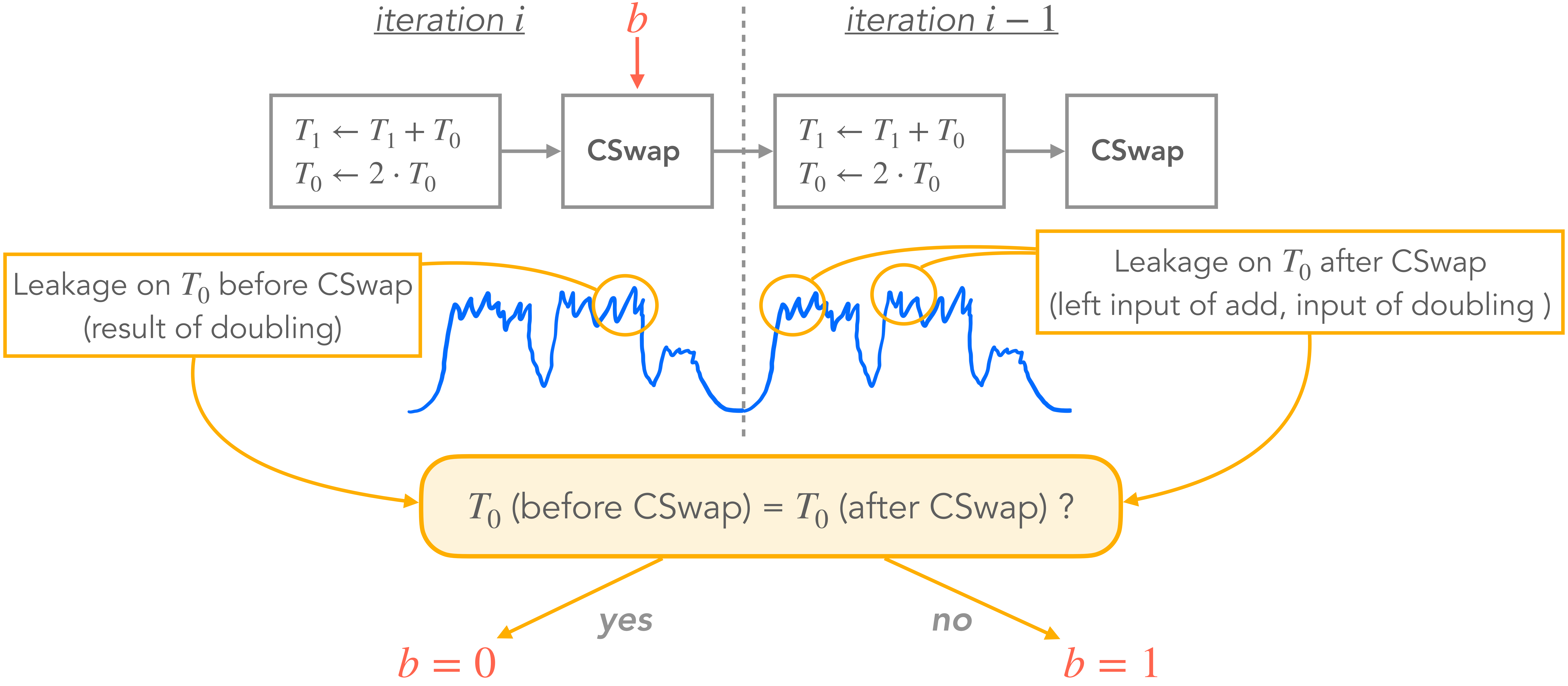
Collision attacks



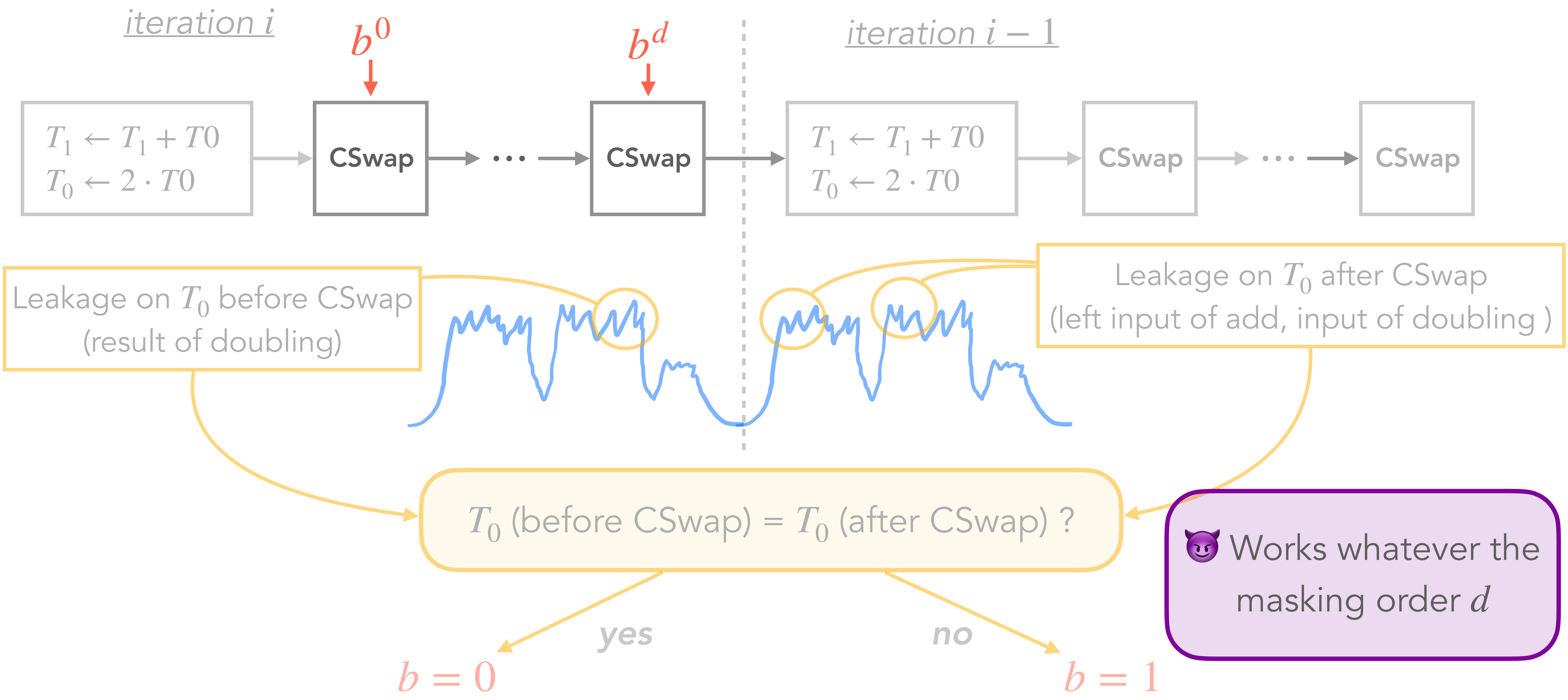
Collision attacks



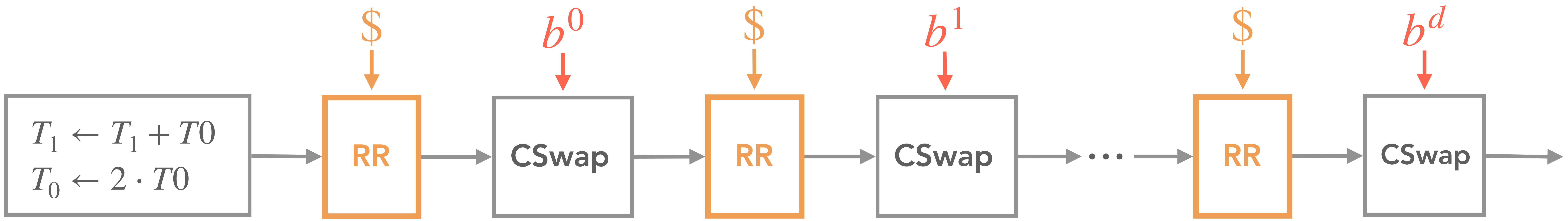
Collision attacks



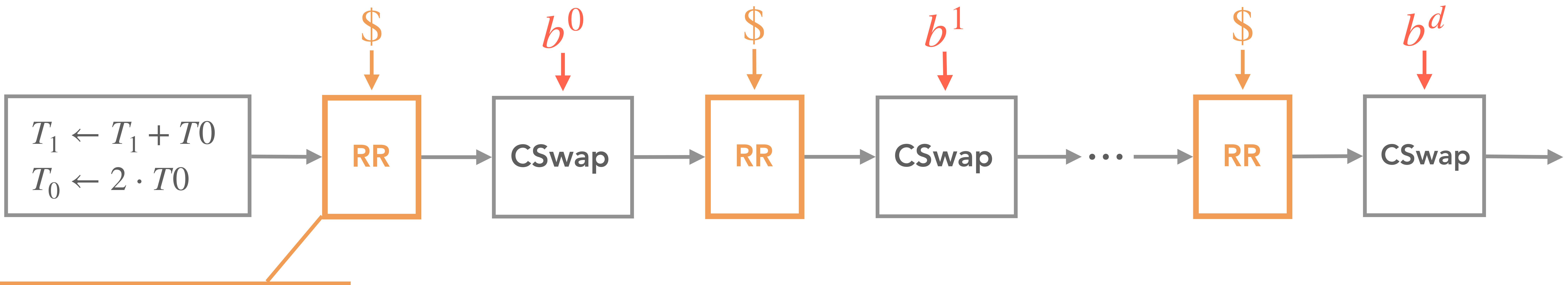
Collision attacks



Our solution



Our solution

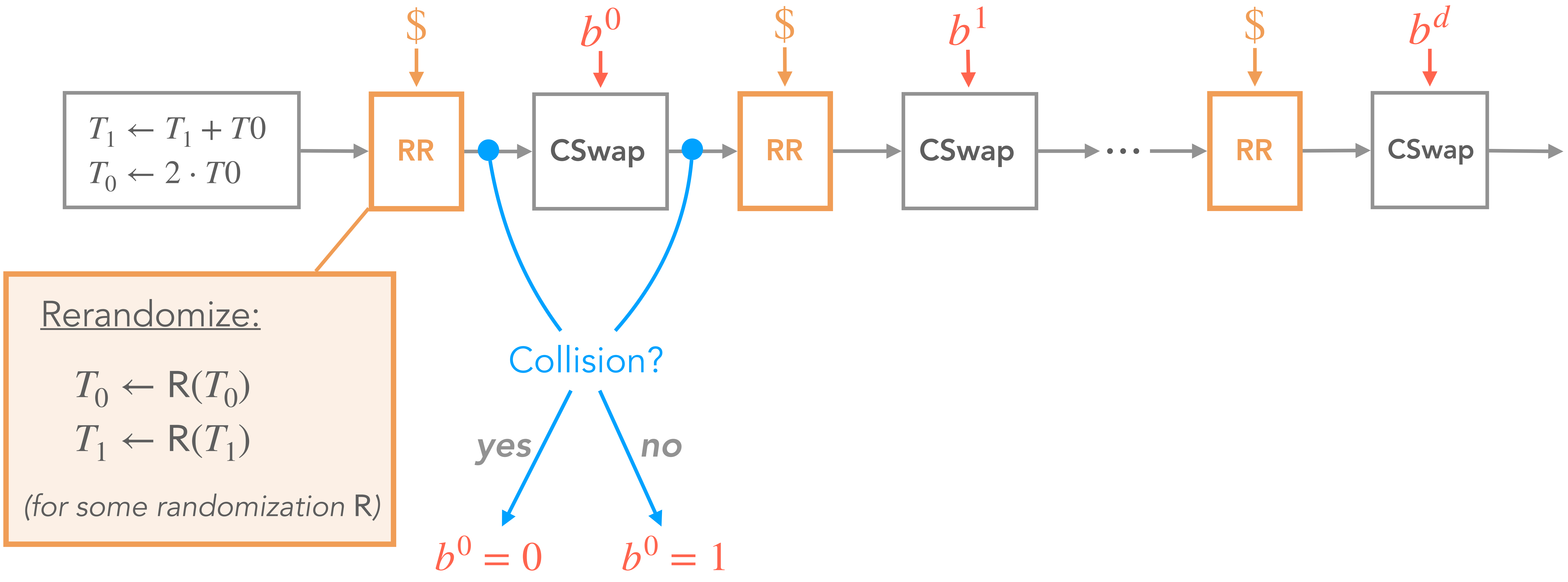


Rerandomize:

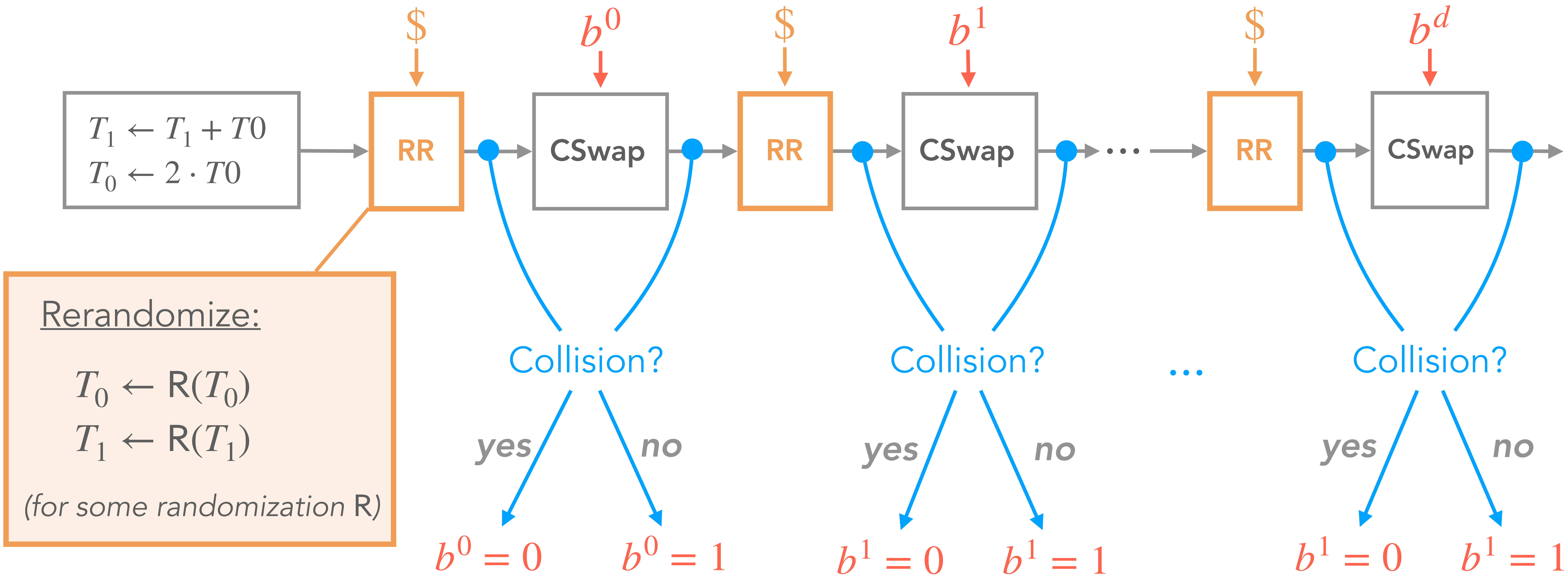
$$T_0 \leftarrow \mathsf{R}(T_0)$$
$$T_1 \leftarrow \mathsf{R}(T_1)$$

(for some randomization R)

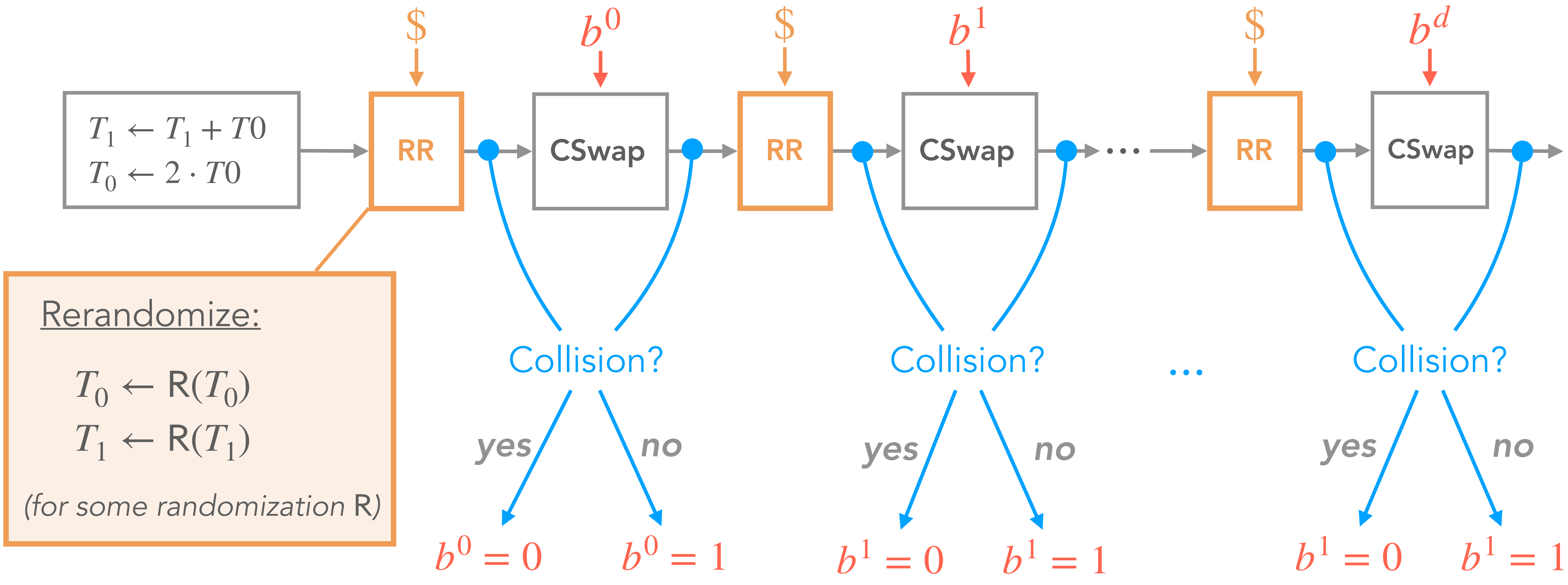
Our solution



Our solution

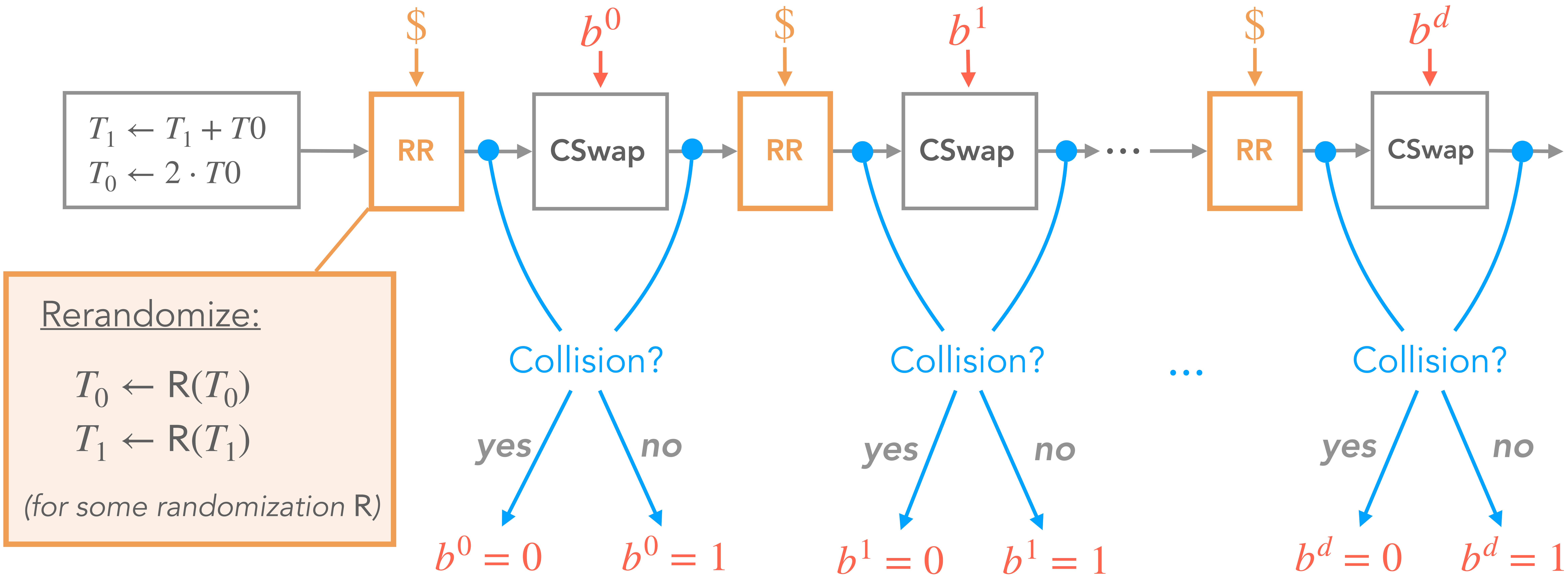


Our solution



😈 Requires collision attack of order $d + 1$

Our solution



👿 Requires collision attack of order $d + 1$

🤔 How to formally prove this high order security?

Formal model

- Computation model: "Randomized Regular Algebraic Program" (RRAP)
  - Two types of variables
    - Algebraic variables $X_1, \ldots X_{\ell_X} \in \mathbb{A}$
    - Index variables $k_1, \ldots, k_{\ell_k} \in \mathbb{Z}$
  - Three types of operations
    - $k_{i_1} \leftarrow \mathsf{op}(k_{i_2}, k_{i_3})$
    - $X_{j_1} \leftarrow \mathsf{Op}(X_{j_2}, X_{j_3})$
    - $X_{j_1} \leftarrow \mathsf{R}(X_{j_1})$

    with $j_1, j_2, j_3 \in \{k_1, \ldots k_{\ell_k}\} \cup \{1, \ldots, \ell_X\}$
- Capture regular algorithms for ECC / RSA / pairings

Formal model

- Computation model: "Randomized Regular Algebraic Program" (RRAP)
  - Two types of variables
    - Algebraic variables $X_1, \ldots X_{\ell_X} \in \mathbb{A}$
    - Index variables $k_1, \ldots, k_{\ell_k} \in \mathbb{Z}$
  - Three types of operations
    - $k_{i_1} \leftarrow \mathsf{op}(k_{i_2}, k_{i_3})$
    - $X_{j_1} \leftarrow \mathsf{Op}(X_{j_2}, X_{j_3})$
    - $X_{j_1} \leftarrow \mathsf{R}(X_{j_1})$

    with $j_1, j_2, j_3 \in \{k_1, \ldots k_{\ell_k}\} \cup \{1, \ldots, \ell_X\}$
- Capture regular algorithms for ECC / RSA / pairings

- Leakage model:
  - Noisy leakage model

Formal model

- <u>Computation model</u>: "Randomized Regular Algebraic Program" (RRAP)
  - Two types of variables
    - Algebraic variables $X_1, \ldots X_{\ell_X} \in \mathbb{A}$
    - Index variables $k_1, \ldots, k_{\ell_k} \in \mathbb{Z}$
  - Three types of operations
    - $\boxed{k_{i_1} \leftarrow \mathsf{op}(k_{i_2}, k_{i_3})}$
    - $\boxed{X_{j_1} \leftarrow \mathsf{Op}(X_{j_2}, X_{j_3})}$
    - $\boxed{X_{j_1} \leftarrow \mathsf{R}(X_{j_1})}$

    with $j_1, j_2, j_3 \in \{k_1, \ldots k_{\ell_k}\} \cup \{1, \ldots, \ell_X\}$
- Capture regular algorithms for ECC / RSA / pairings

- <u>Leakage model:</u>
  - Noisy leakage model

    → *Leaks* $f(k_{i_2}, k_{i_3})$

    → *Leaks* $f(j_1, j_2, j_3, X_{j_2}, X_{j_3})$

    → *Leaks* $f(j_1, X_{j_1}, \mathsf{R}(X_{j_1}))$

    with f a *δ-noisy leakage function*:

$$\mathrm{SD}\big(U; (U \mid f(U)\big) \leq \delta$$

Formal model

- Computation model: "Randomized Regular Algebraic Program" (RRAP)
  - Two types of variables
    - Algebraic variables $X_1, \ldots X_{\ell_X} \in \mathbb{A}$
    - Index variables $k_1, \ldots, k_{\ell_k} \in \mathbb{Z}$
  - Three types of operations
    - $\boxed{k_{i_1} \leftarrow \mathsf{op}(k_{i_2}, k_{i_3})}$
    - $\boxed{X_{j_1} \leftarrow \mathsf{Op}(X_{j_2}, X_{j_3})}$
    - $\boxed{X_{j_1} \leftarrow \mathsf{R}(X_{j_1})}$

    with $j_1, j_2, j_3 \in \{k_1, \ldots k_{\ell_k}\} \cup \{1, \ldots, \ell_X\}$
- Capture regular algorithms for ECC / RSA / pairings

- Leakage model:
  - Noisy leakage model

    Leaks $f(k_{i_2}, k_{i_3})$

    Leaks $f(j_1, j_2, j_3, X_{j_2}, X_{j_3})$

    Leaks $f(j_1, X_{j_1}, \mathsf{R}(X_{j_1}))$

    with f a *δ-noisy leakage function*:

$$\mathrm{SD}\big(U;\ (U \mid f(U)\big) \leq \delta$$

  - Hiddenness assumption

💡 Capture that $x \mapsto f \circ \mathsf{R}(x)$ hides the information on x

Hiddenness assumption

📜 **Hiddenness assumption**
(simple version)

- Let f a (noisy) leakage function
- Let $\mathsf{R} : \mathbb{A} \rightarrow \mathbb{A}$ a rand. operation
- The pair (f, R) is ε-*hiding* if

$$\forall x : \; f(\mathsf{R}(x)) \approx_{\varepsilon} f(U)$$

with U uniform r.v. over $\mathbb{A}$

Hiddenness assumption

📜 **Hiddenness assumption**
(simple version)

- Let f a (noisy) leakage function
- Let $\mathsf{R} : \mathbb{A} \to \mathbb{A}$ a rand. operation
- The pair (f, R) is *ε-hiding* if

$$\forall x : \; f(\mathsf{R}(x)) \approx_\varepsilon f(U)$$

with U uniform r.v. over $\mathbb{A}$

Complete version: adapted to multiple muti-input operations

Hiddenness assumption

Hiddenness assumption
(simple version)

- Let f a (noisy) leakage function
- Let $\mathsf{R} : \mathbb{A} \to \mathbb{A}$ a rand. operation
- The pair (f, R) is ε-*hiding* if

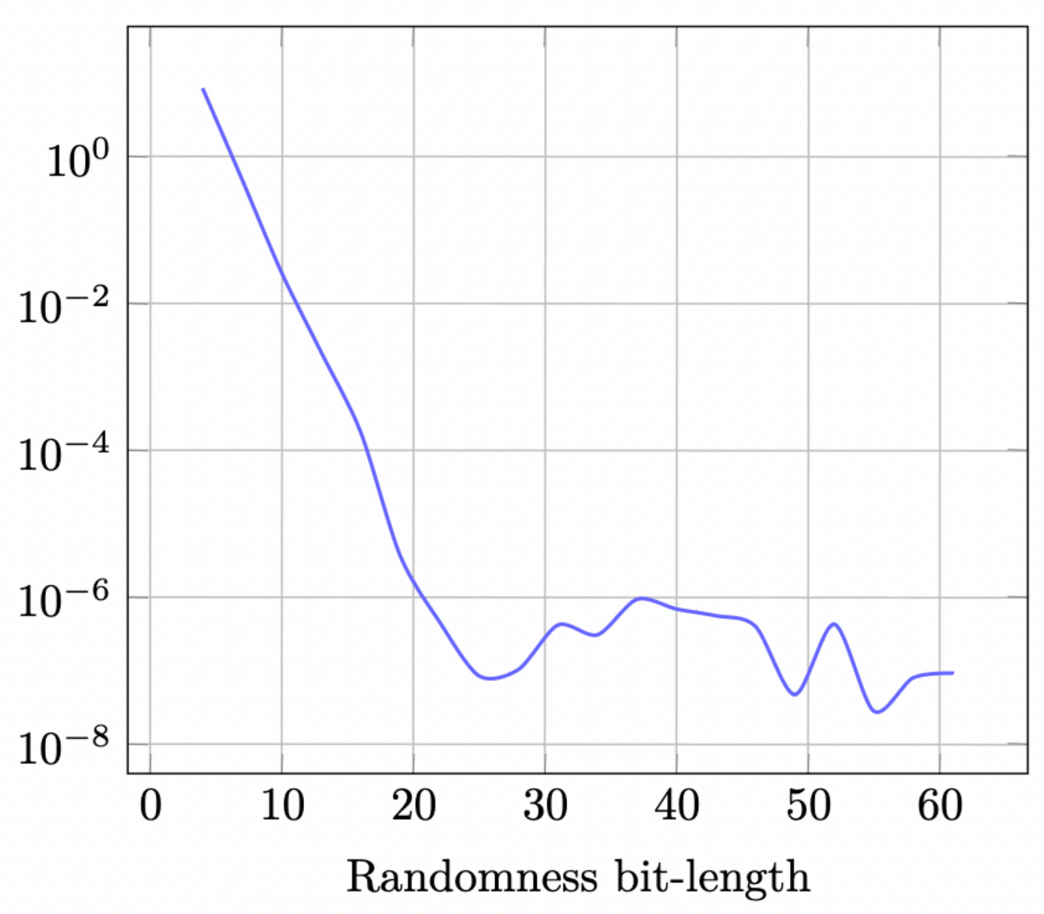
$$\forall x : \ f(\mathsf{R}(x)) \approx_{\varepsilon} f(U)$$

with U uniform r.v. over $\mathbb{A}$

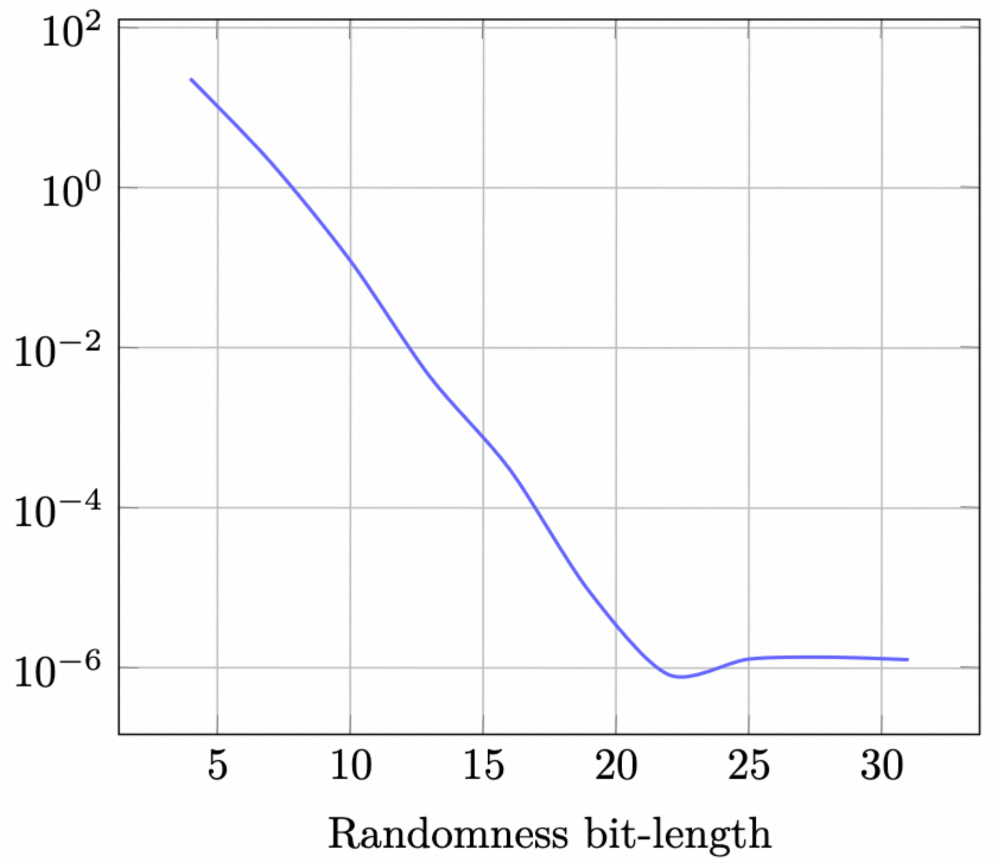
Complete version: adapted to multiple muti-input operations

Experiments:

KL divergence between $f(\mathsf{R}(x))$ and $f(U)$
(Hamming weight + Gaussian noise model)



R = field element randomization



R = randomization of projective coord.

Hiddenness assumption

Hiddenness assumption
(simple version)

- Let f a (noisy) leakage function
- Let $\mathsf{R} : \mathbb{A} \to \mathbb{A}$ a rand. operation
- The pair (f, R) is ε-*hiding* if

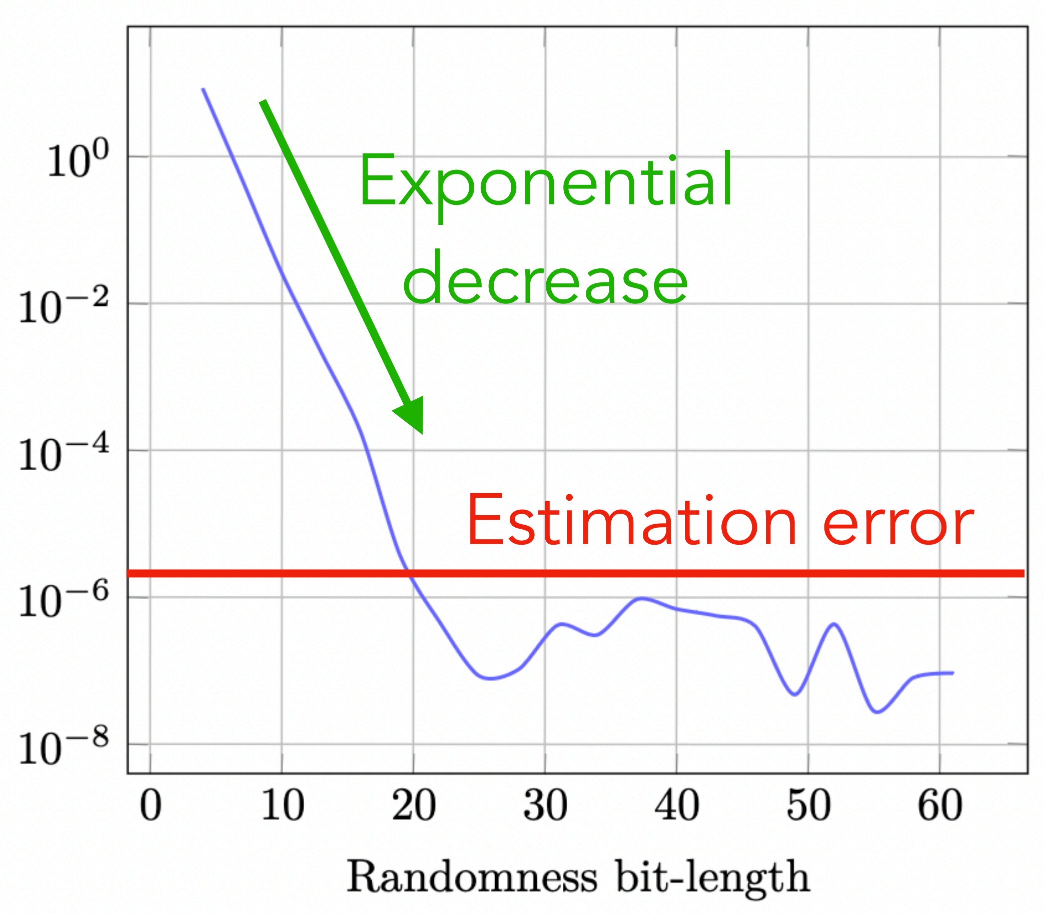
$$\forall x : \; f(\mathsf{R}(x)) \approx_{\varepsilon} f(U)$$

with U uniform r.v. over $\mathbb{A}$

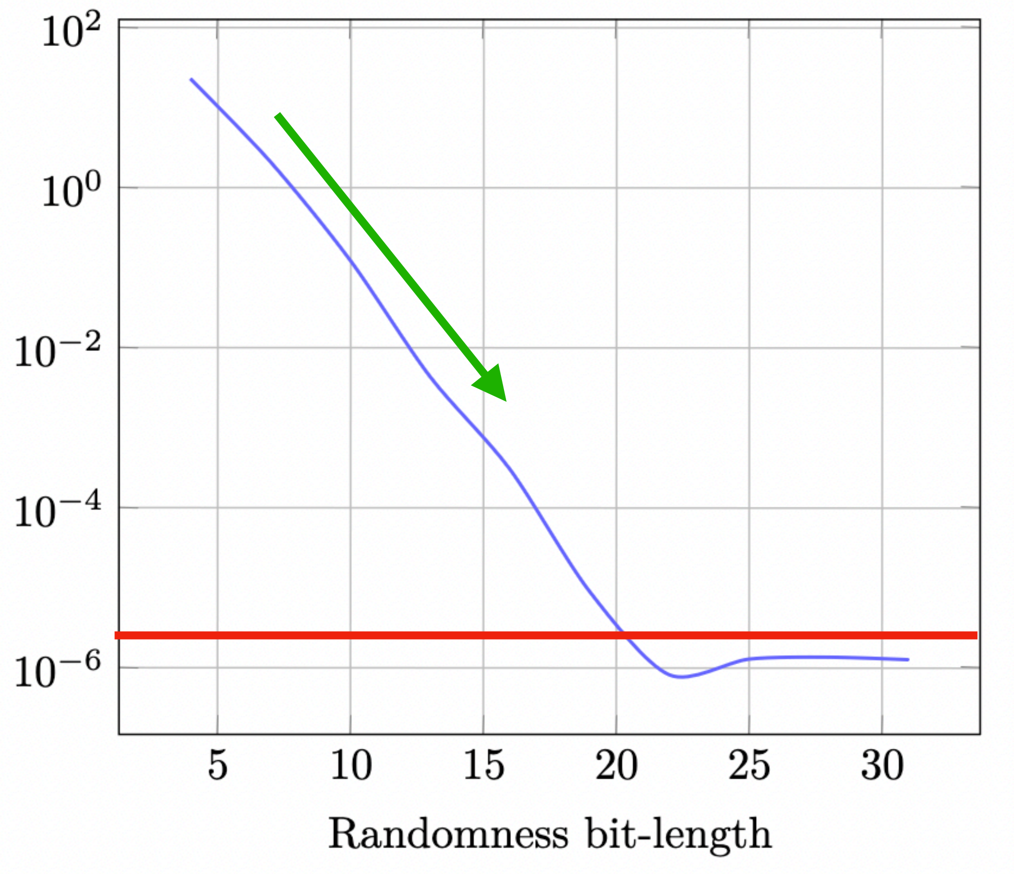
Complete version: adapted to multiple muti-input operations

Experiments:

KL divergence between $f(\mathsf{R}(x))$ and $f(U)$ (Hamming weight + Gaussian noise model)



R = field element randomization



R = randomization of projective coord.

Security proof

Leakage resilience:

A RRAP is γ-leakage resilient if $\exists$ a simulator s.t. Sim( ) $\approx_\gamma$ Leak($\overrightarrow{k}$)

Security proof

Leakage resilience:

A RRAP is γ-leakage resilient if $\exists$ a simulator s.t. Sim( ) $\approx_\gamma$ Leak($\overrightarrow{k}$)

Security theorem:

Our generic countermeasure is γ-leakage resilient with:

$$\gamma \leq (cst_1 \cdot \delta)^{d+1} + cst_2 \cdot \varepsilon$$

Security proof

Leakage resilience:

A RRAP is γ-leakage resilient if $\exists$ a simulator s.t. Sim( ) $\approx_\gamma$ Leak($\overrightarrow{k}$)

Security theorem:

Our generic countermeasure is γ-leakage resilient with:

$$\gamma \leq (cst_1 \cdot \delta)^{d+1} + cst_2 \cdot \varepsilon$$

δ-noisy leakage functions

masking order d

ε-hiddenness

Security proof

Leakage resilience:

A RRAP is γ-leakage resilient if $\exists$ a simulator s.t. Sim( ) $\approx_\gamma$ Leak($\overrightarrow{k}$)

Security theorem:

Our generic countermeasure is γ-leakage resilient with:

$$\gamma \leq (cst_1 \cdot \delta)^{d+1} + cst_2 \cdot \varepsilon$$

δ-noisy leakage functions

masking order d

ε-hiddenness

constants related to # operations

Security proof

Leakage resilience:

A RRAP is γ-leakage resilient if $\exists$ a simulator s.t. Sim( ) $\approx_\gamma$ Leak($\overrightarrow{k}$)

Security theorem:

Our generic countermeasure is γ-leakage resilient with:

$$\gamma \leq (cst_1 \cdot \delta)^{d+1} + cst_2 \; \varepsilon$$

δ-noisy leakage functions

masking order d

ε-hiddenness

constants related to # operations

Proof sketch:

1. Apply ε-hiddenness to replace re-randomized variables by new uniform variables

   $\rightarrow cst_2 \cdot \varepsilon$ gap

2. Replace noisy leakage by random probing leakage

   $\rightarrow$ no gap

   $\rightarrow (cst_1 \cdot \delta)^{d+1}$ probability of simulation failure

Application

- Generic algorithm applicable to any RRAP
- Several ECC scalar mult. algorithms expressed in our framework:
  - Montgomery ladder (point level & coordinate level)
  - Joye ladder
  - Signed binary ladder
  - Fixed-window scalar multiplication
- PoC smart card implementation
  - (signed binary ladder with XY-only co-Z coordinates)

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

Jacobian coordinate randomization

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

Jacobian coordinate randomization

Double randomization

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

Jacobian coordinate randomization

Double randomization

$[k]P = [k_0]P + \cdots + [k_d]P$

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

Jacobian coordinate randomization

Double randomization

$[k]P = [k_0]P + \cdots + [k_d]P$

ISW applied to all mult.

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Performance estimations

	order 1	order 2	order 4	order 8
Our countermeasure (overhead)				
R_1 - $h = 32$	1,35	1,39	1,47	1,64
R_1 - $h = 64$	1,73	1,81	1,98	2,31
R_1 - $h = 128$	2,58	2,75	3,08	3,75
R_2	3	4	6	10
R_1 & R_2 - $h = 32$	5,48	7,59	11,81	20,25
R_1 & R_2 - $h = 64$	6,77	9,38	14,58	25
R_1 & R_2 - $h = 128$	9,75	13,5	21	36
Other countermeasures (overhead)				
scalar splitting	2	3	5	9
naive ISW	4	9	25	81

Field element randomization

Jacobian coordinate randomization

Double randomization

$[k]P = [k_0]P + \cdots + [k_d]P$

ISW applied to all mult.

Less secure than our solution (provided that hiddenness holds)

* Assume 12 multiplications per loop iteration

** Neglect add / sub vs. multiplications

Conclusion

- Formal model for regular exponentiation-like algorithms (with randomization)
- Formalisation of the hiddenness assumption
- Generic provably secure countermeasure
- Application to several ECC scalar mult. algorithms
- **Perspectives**:
    - Challenge the hiddenness assumption in practice
    - Applications to other algorithms / randomization techniques
    - Practical implementations and attacks