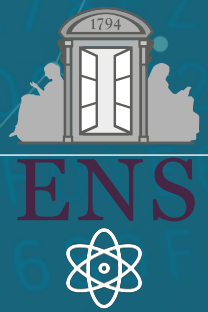
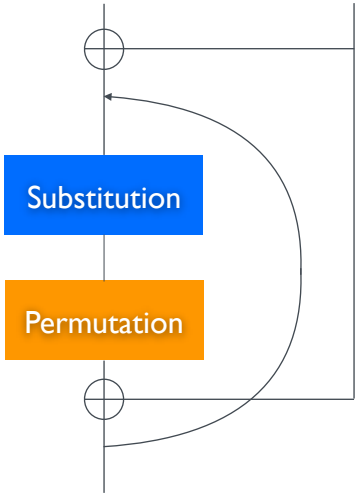
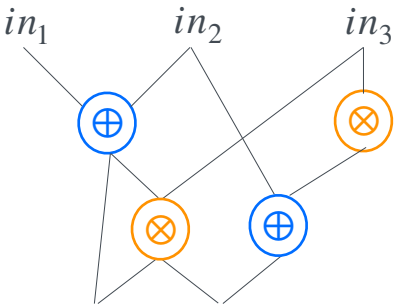


How to Securely Compute with Noisy Leakage in Quasilinear Complexity

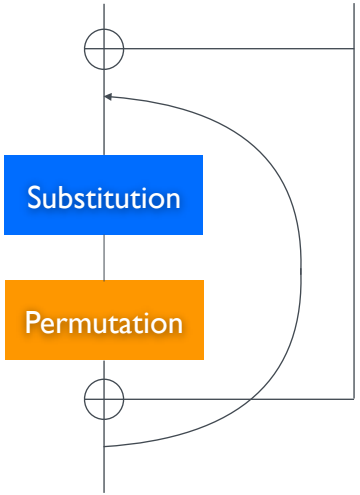
Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain
dahmun.goudarzi@pqshield.com

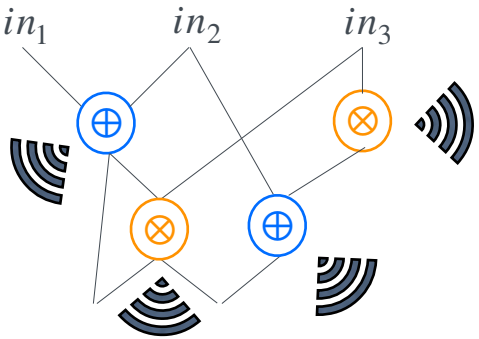
Side-Channel Attacks





Side-Channel Attacks



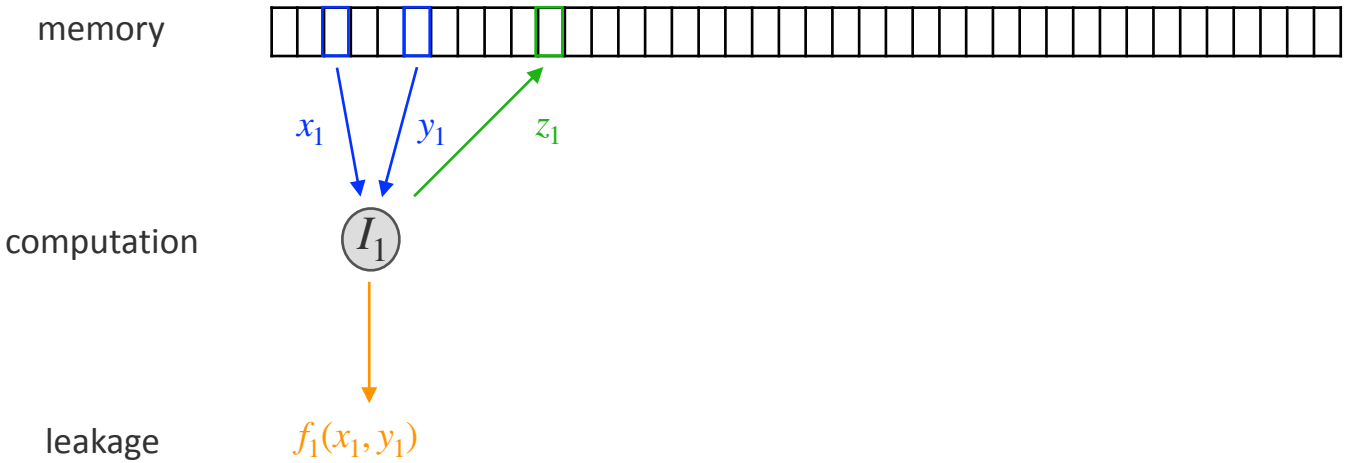


Leakage depends on the key!

Only Computation Leaks

- Micali, Reyzin. *Physically Observable Cryptography* (TCC 2004)
- Key assumption: **only computation leaks information**

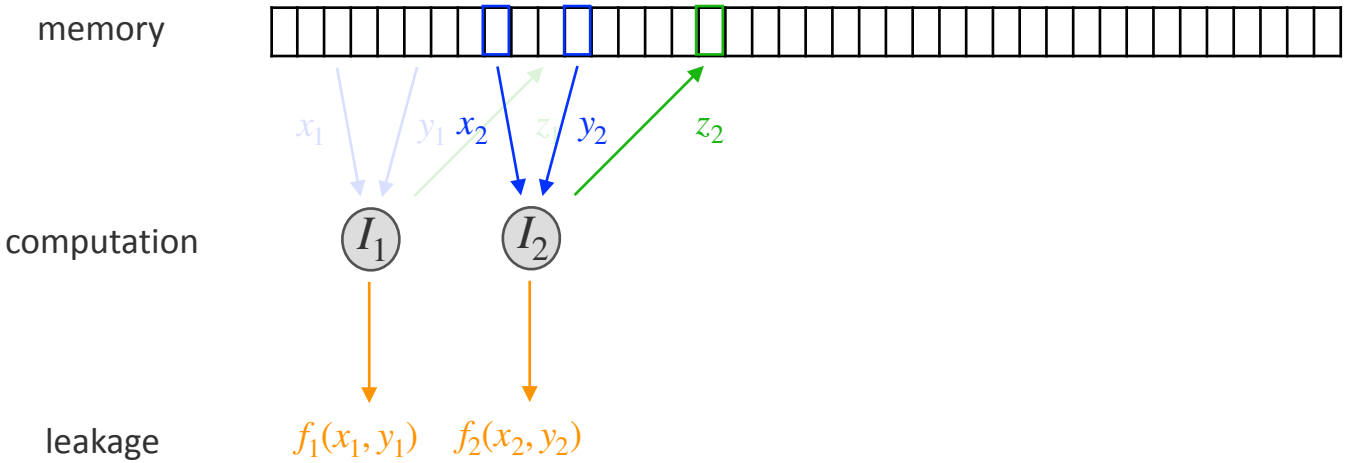
  Computation divided in sub-computations $I_1, I_2, \ldots, I_s$



Only Computation Leaks

- Micali, Reyzin. *Physically Observable Cryptography* (TCC 2004)
- Key assumption: **only computation leaks information**

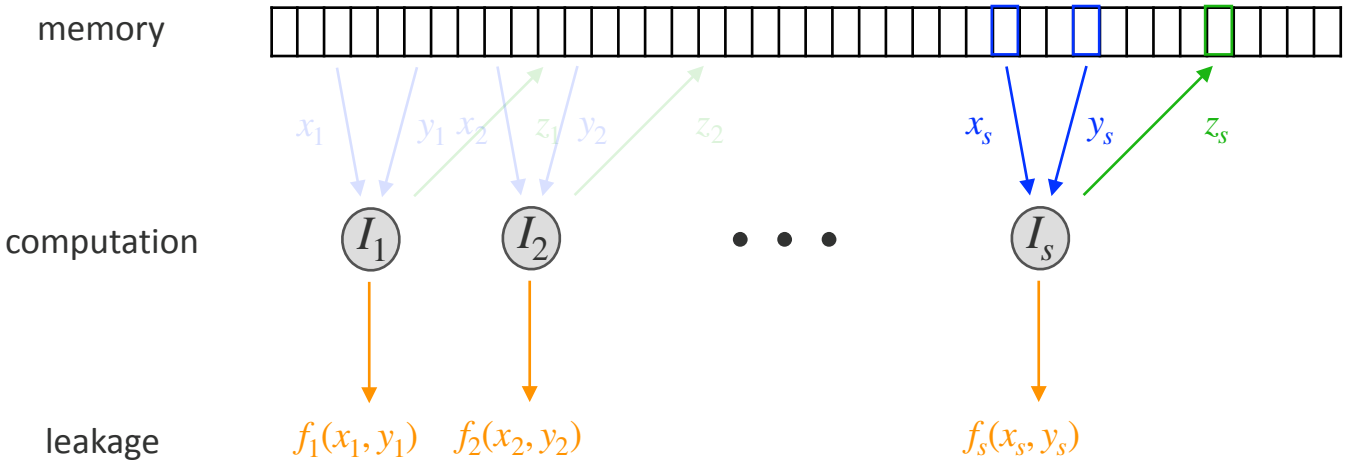
  Computation divided in sub-computations $I_1, I_2, \ldots, I_s$



Only Computation Leaks

- Micali, Reyzin. *Physically Observable Cryptography* (TCC 2004)
- Key assumption: **only computation leaks information**

  Computation divided in sub-computations $I_1, I_2, \ldots, I_s$

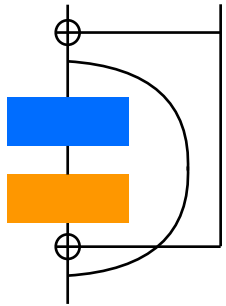


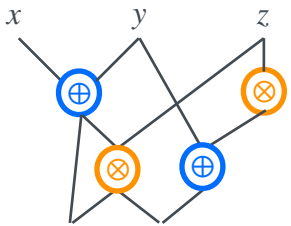
Noisy Leakage Model

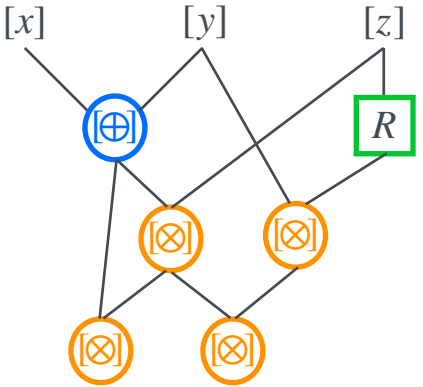
- Prouff, Rivain. *Masking against Side-Channel Attacks: A Formal Security Proof* (Eurocrypt 2013)
- f is a non-deterministic function: $f(x) = f(x, randomness)$
- Informally: An observation $f(X)$ introduces a **bounded bias** δ in the distribution of X
- Statistical distance: $\Delta\left(X;\ (X \mid f(X)\right) \leq \delta \Rightarrow f$ is δ-noisy
- Capture **any** noisy leakage distribution (single parameter δ)

no information $0 \leq \delta \leq 1$ full information

Implementation Transformation







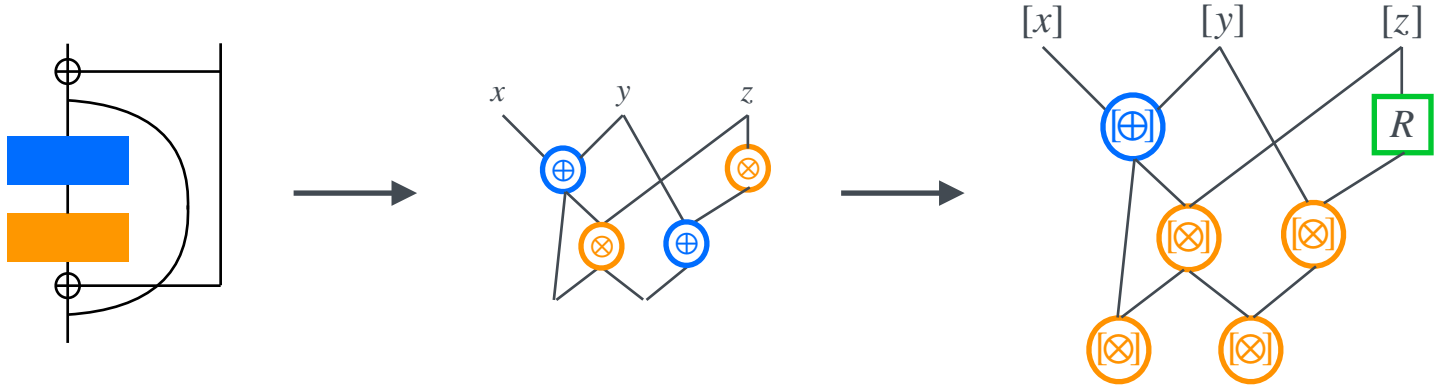
$[x]$: encoding

$[\oplus]$: addition gadget

$[\otimes]$: multiplication gadget

R : refresh gadget

Implementation Transformation



$[x]$: encoding

$[\oplus]$: addition gadget

$[\otimes]$: multiplication gadget

R : refresh gadget

Goal:

1. Minimal circuit complexity
2. Constant leakage rate

Leakage Rate

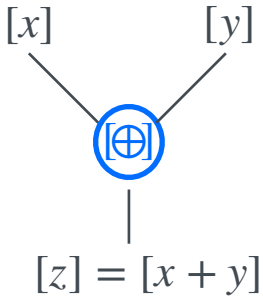
- Leakage rate: #leaked information / circuit complexity
- Importance of leakage rate (example from Andrychowicz *et al.* Circuit Compilers with O(1/log n) Leakage Rage. EC 16)

Circuit 1	Circuit 2
• Size: 1k gates	• Size: 1M gates
• Tolerate leakage: 10 wires	• Tolerate leakage: 100 wires
• Leakage rate: 1% of the wires	• Leakage rate: 0.01% of the wires

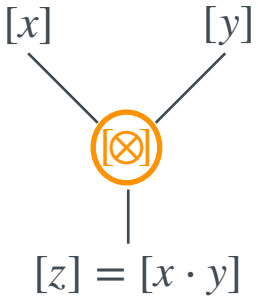
Classical Approach

- Encoding: $[x] = (x_1, x_2, \ldots, x_d)$ s.t $x = \sum_{i=1}^{d} x_i$

Addition gadget



Multiplication gadget



Refresh gadget

- ISW-mult: $\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} \rightarrow \begin{pmatrix} a_1b_1 & a_1b_2 \oplus a_2b_1 & a_1b_3 \oplus a_3b_1 \\ 0 & a_2b_2 & a_2b_3 \oplus a_3b_2 \\ 0 & 0 & a_3b_3 \end{pmatrix} \rightarrow \begin{pmatrix} a_1b_1 & (a_1b_2 \oplus r_{1,2}) \oplus a_2b_1 & (a_1b_3 \oplus r_{1,3}) \oplus a_3b_1 \\ r_{1,2} & a_2b_2 & (a_2b_3 \oplus r_{2,3}) \oplus a_3b_2 \\ r_{1,3} & r_{2,3} & a_3b_3 \end{pmatrix}$

Limitations

- Complexity: $O(d^2)$

- Leakage rate: $O(1/d)$

$$\begin{pmatrix} a_1b_1 & a_1b_2 & \dots & a_1b_d \\ a_2b_1 & a_2b_2 & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_db_1 & a_db_2 & \dots & a_db_d \end{pmatrix}$$

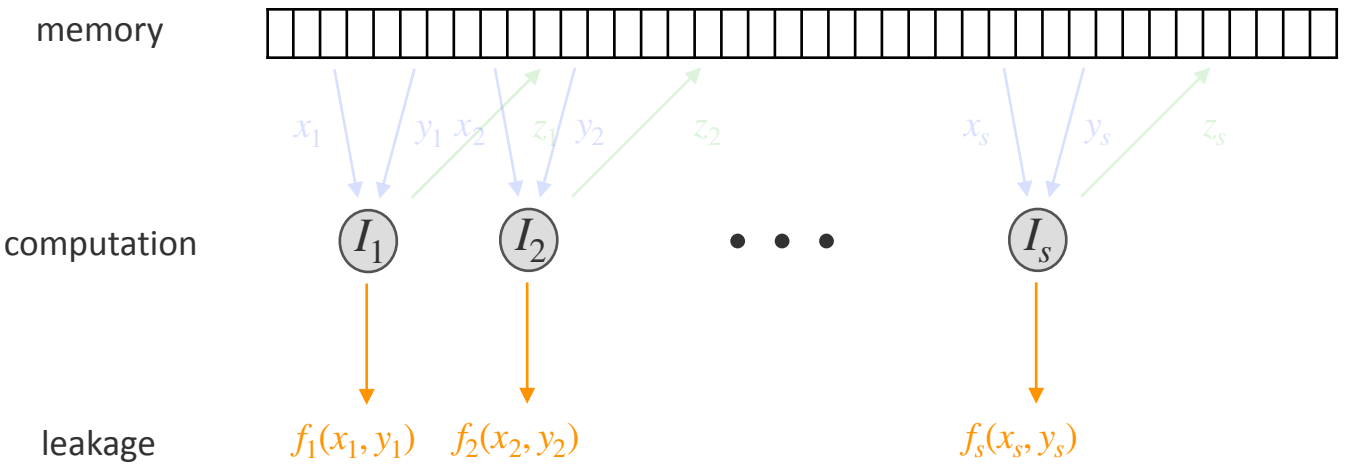
each share is manipulated d times

Our Results

- Asymptotic comparison of secure schemes in the noisy leakage model.

	ISW	ADF	This talk
Leakage rate	$O(1/d)$	$O(1)$	$\tilde{O}(1)$
Complexity blow-up	$O(d^2)$	$\tilde{O}(d^2)$	$\tilde{O}(d)$

Random Probing Model



- The leakage function f is such that

$$f(x_i, y_i) = \begin{cases} (x_i, y_i) & \text{with probability } \epsilon \\ \perp & \text{with probability } 1 - \epsilon \end{cases}$$

Random Probing to Noisy Leakage Security

- Duc, Dziembowski, Faust. *Unifying Leakage Models: from Probing Attacks to Noisy Leakage* (Eurocrypt 2014)

- **Key lemma**

  Every δ-noisy function f can be written as

  $$f(\cdot) = f' \circ \varphi(\cdot)$$

  where φ is an ϵ-random probing function with $\epsilon = \Theta(\delta)$

- ϵ-random probing security $\Rightarrow$ δ-noisy leakage security with $\delta = \Theta(\epsilon)$

Our Encoding

- A variable $a \in \mathbb{F}$ is randomly encoded as

$$\text{Enc}_\omega(a) = (a_0, a_1, \ldots, a_{d-1}) \text{ where } \sum_{i=0}^{d-1} a_i \cdot \omega^i = a$$

- ω is random in $\mathbb{F}^*$ but can be leaked to the adversary

- Encoding linearity

$$\text{Enc}(a) + \text{Enc}(b) = \text{Enc}(a + b)$$

Multiplying Encodings

- Goal: Compute $\mathrm{Enc}(a \cdot b)$ from $\mathrm{Enc}(a)$ and $\mathrm{Enc}(b)$
- Consider

$$P_a(X) = \sum_{i=0}^{d-1} a_i \cdot X^i \text{ and } P_b(X) = \sum_{i=0}^{d-1} b_i \cdot X^i$$

- By definition $P_a(\omega) = a$ and $P_b(\omega) = b$
- Define

$$P_t(X) = P_a(X) \cdot P_b(X) = \sum_{i=0}^{2d-1} t_i \cdot X^i$$

- We have $P_t(\omega) = a \cdot b$ but $\deg(P_t) = 2d - 1 \geq d - 1$

Multiplying Encodings

- Compression procedure:

$$a \cdot b = P_t(\omega) = \sum_{i=0}^{2d-1} t_i \cdot \omega^i$$

$$a \cdot b = P_c(\omega) = \sum_{i=0}^{d-1} c_i \cdot \omega^i$$

with $c_i = t_i + t_{d+i} \cdot \omega^n$

- $\text{Enc}(a \cdot b) = (c_0, c_1, \ldots, c_{d-1})$

Number Theoretic Transform (NTT)

- Compute $P_t(X) = P_a(X) \cdot P_b(X)$ in $O(d \log d)$
- Evaluate P_a and P_b in $2d$ points $\alpha, \beta, \ldots, \gamma$

$$\begin{pmatrix} \alpha^0 & \alpha^1 & \ldots & \alpha^{d-1} \\ \beta^0 & \beta^1 & \ldots & \beta^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma^0 & \gamma^1 & \ldots & \gamma^{d-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} P_a(\alpha) \\ P_a(\beta) \\ \vdots \\ P_a(\gamma) \end{pmatrix}$$

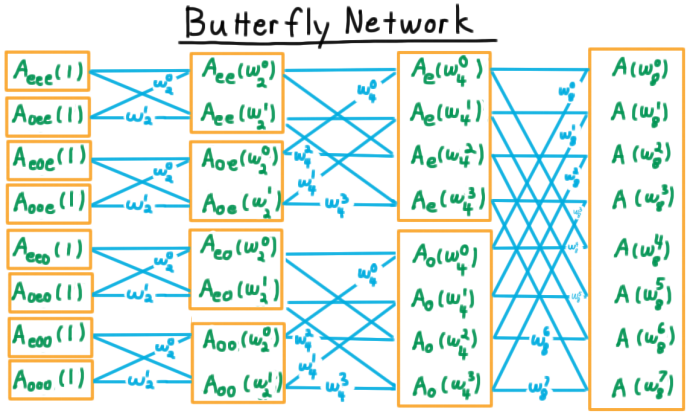
- Get the corresponding evaluations

$$P_t(\alpha) = P_a(\alpha) \cdot P_b(\alpha), \ldots$$

- Integrate the coefficients of P_t from the $2d$ evaluations (multiplication by the inverse matrix)

Number Theoretic Transform (NTT)

- Takes points $\alpha, \beta, \ldots, \gamma$ as the $2d$-th roots of unity
- Each matrix multiplication computed as a **butterfly network**



- $(\log 2d)$ steps of d butterfly operations $\Rightarrow O(d \log d)$
- Constraint: $2d$-th roots of unity $\in \mathbb{F}$

Random Probing Security

- Consider the NTT:

$$\begin{pmatrix} \alpha^0 & \alpha^1 & \dots & \alpha^{d-1} \\ \beta^0 & \beta^1 & \dots & \beta^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma^0 & \gamma^1 & \dots & \gamma^{d-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} = \begin{pmatrix} P_a(\alpha) \\ P_a(\beta) \\ \vdots \\ P_a(\gamma) \end{pmatrix}$$

- ϵ-random probing model
  - $t \approx O(\epsilon\, d \log d)$ intermediate variables leak
- Leakage: $M \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix}$ for some $(t \times d)$ matrix M

Random Probing Security

- Recall: $a = (\omega^0, \omega^1, \ldots, \omega^{d-1}) \cdot (a_0, a_1, \ldots, a_{d-1})^T$

- Lemma 1:

$$M \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{pmatrix} \text{ reveals info on } a \quad \Leftrightarrow \quad (\omega^0, \omega^1, \ldots, \omega^{d-1}) \in \langle M \rangle$$

- Lemma 2:

$$\text{If } t < d, P[(\omega^0, \omega^1, \ldots, \omega^{d-1}) \in \langle M \rangle] \leq d/|\mathbb{F}|$$

- Conditions:
  - $t < d$ and $t = \epsilon d \log d \quad \Leftarrow \quad \epsilon = O(1/\log d)$
  - $d/|F| \leq 2^{-\lambda} \quad \Leftarrow \quad |p| = \log d + \lambda$

Arithmetic Program Compiler

- **Copy gadget:** $(a'_0, a'_1 \ldots, a'_{d-1}) \leftarrow \text{refresh}(a_0, a_1, \ldots, a_{d-1})$

- **Addition gadget:** $(c_0, c_1 \ldots, c_{d-1}) \leftarrow \text{refresh}(a_0 + b_0, a_1 + b_1, \ldots, a_{d-1} + b_{d-1})$

- **Multiplication gadget:** $(c_0, c_1 \ldots, c_{d-1}) \leftarrow \text{refresh}(\text{NTTMult}((a_0, a_1, \ldots, a_{d-1}), (b_0, b_1, \ldots, b_{d-1}))$

  where NTTMult is composed of five steps:

$$
\begin{aligned}
(u_0, u_1, \ldots, u_{2d-1}) &\leftarrow \text{NTT}_\zeta(a_0, a_1, \ldots, a_{d-1}, 0, \ldots, 0) \\
(r_0, r_1, \ldots, r_{2d-1}) &\leftarrow \text{NTT}_\zeta(b_0, b_1, \ldots, b_{d-1}, 0, \ldots, 0) \\
(s_0, s_1, \ldots, s_{2d-1}) &\leftarrow (2d)^{-1}(u_0 \cdot r_0, u_1 \cdot r_1, \ldots, u_{2d-1} \cdot r_{2d-1}) \\
(t_0, t_1, \ldots, t_{2d-1}) &\leftarrow \text{NTT}_{\zeta^{-1}}(s_0, s_1, \ldots, s_{2d-1}) \\
(c_0, c_1, \ldots, c_{d-1}) &\leftarrow \text{compress}(t_0, t_1, \ldots, t_{2d-1})
\end{aligned}
$$

Refresh Procedure

- 1. Sample a random ω-encoding $(r_0, r_1, \ldots, r_{d-1}) \leftarrow \mathrm{Enc}_\omega(0)$

  2. Set $a_i' = a_i + r_i$ for $i = 0$ to $d - 1$

- Sampling encodings of 0:

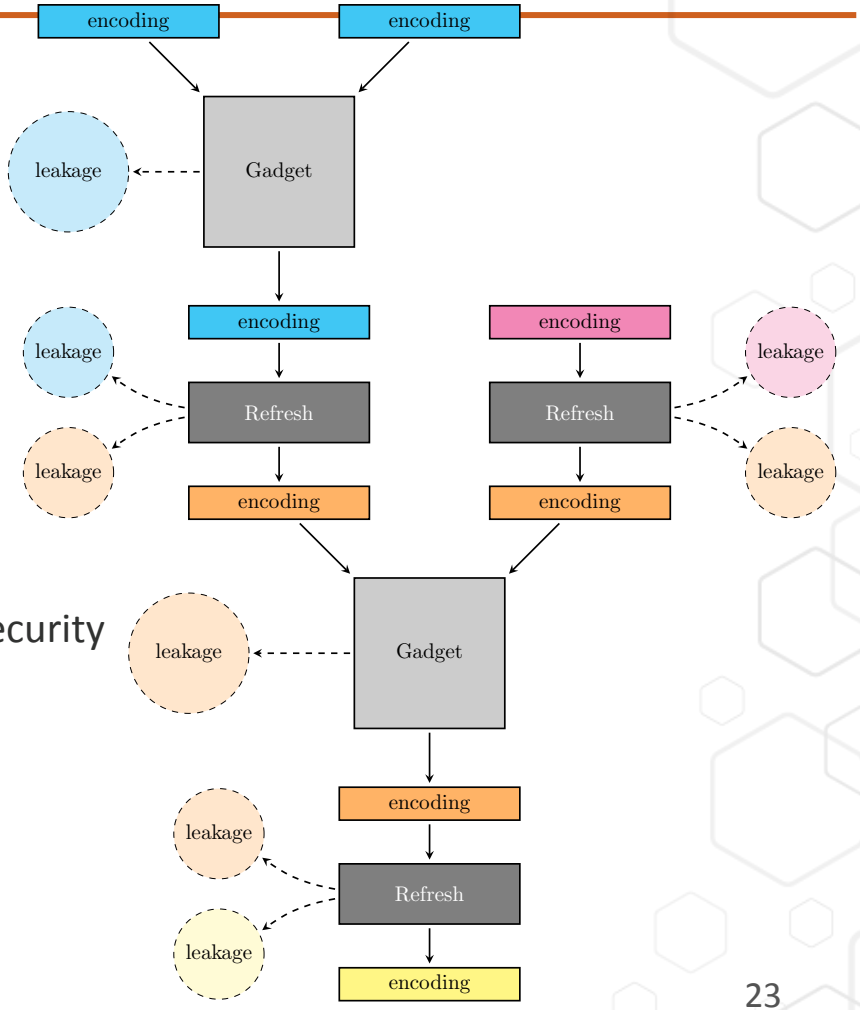
  1. Pick $d - 1$ random values $u_0, u_1, \ldots, u_{d-2}$ over $\mathbb{F}$

  2. Output $(r_i)_{i=0}^{d-1} = \mathrm{NTTMult}((u_0, u_1, \ldots, u_{d-2}, 0), (e_0, e_1, \ldots, e_{d-1}))$

- Used to ensure an encoding is not used in more than 2 gadgets

Composition

- Key properties
  - uniformity
  - I/O separability



- DDF14 reduction:

  ϵ-random probing security $\Rightarrow$ δ-noisy leakage security

  with $\delta = O(\epsilon) = O(1/\log d)$

Conclusion

- New circuit compiler secure in the noisy leakage model with

	ISW	ADF	This talk
Leakage rate	$O(1/d)$	$O(1)$	$\tilde{O}(1)$
Complexity blow-up	$O(d^2)$	$\tilde{O}(d^2)$	$\tilde{O}(d)$

- Shamir secret sharing + NTT

PQ SHIELD

Thank you!

Questions?