On non-uniformity in threshold schemes

Joan Daemen

STMicroelectronics

Radboud University

SPACE 2016
Hyderabad, December 17, 2016

Outline

1 The original motivation

2 Distributions, spectrum and collision probability

3 Mappings and correlation matrices

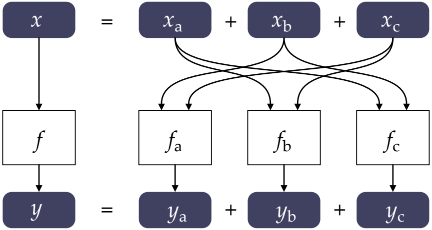
4 In the setting of a shared implementation

5 Achieving uniformity

Protection of KECCAK against first-order DPA

- KEYAK, KETJE and keyed KECCAK in the field
  - Banking cards, ID, public transport, *secure elements*, ...
  - Protection against side-channels is relevant: DPA, DEMA
- Threshold scheme [Nikova, Rijmen, Schläffer]
  - represent (native) state bits as n shares
  - for KECCAK: 3 shares
  - Incompleteness: each combinatorial block only takes 2 shares
  - If input uniformly shared, computation uncorrelated to native state
  - *Provably secure* against 1st order DPA/DEMA

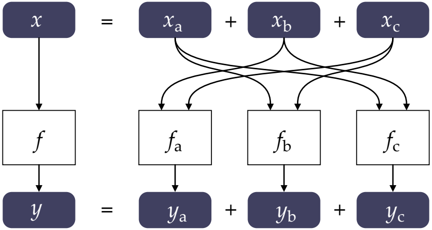
A 3-share threshold implementation



- Uniformly shared variable x:
  - All (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable
  - Equivalently: $\forall x : (x_b, x_c)$ uniform
  - (x_b, x_c): *randomization vector*
- (f_a, f_b, f_c) is sharing of f
- Given x, mapping from (x_b, x_c) to (y_b, y_c) is deterministic:

$$(y_b, y_c) = (f_b(x + x_b + x_c, x_c), f_c(x + x_b + x_c, x_b))$$

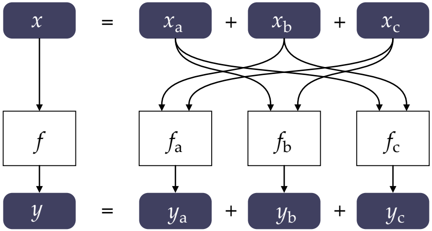
A 3-share threshold implementation



- Uniformly shared variable x:
  - All (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable
  - Equivalently: $\forall x : (x_b, x_c)$ uniform
  - (x_b, x_c): *randomization vector*
- (f_a, f_b, f_c) is sharing of f
- Given x, mapping from (x_b, x_c) to (y_b, y_c) is deterministic:

$$(y_b, y_c) = (f_b(x + x_b + x_c, x_c), f_c(x + x_b + x_c, x_b))$$

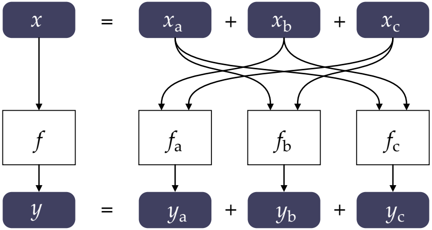
A 3-share threshold implementation



- Uniformly shared variable x:
  - All (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable
  - Equivalently: $\forall x : (x_b, x_c)$ uniform
  - (x_b, x_c): *randomization vector*
- (f_a, f_b, f_c) is sharing of f
- Given x, mapping from (x_b, x_c) to (y_b, y_c) is deterministic:

$$(y_b, y_c) = (f_b(x + x_b + x_c, x_c), f_c(x + x_b + x_c, x_b))$$

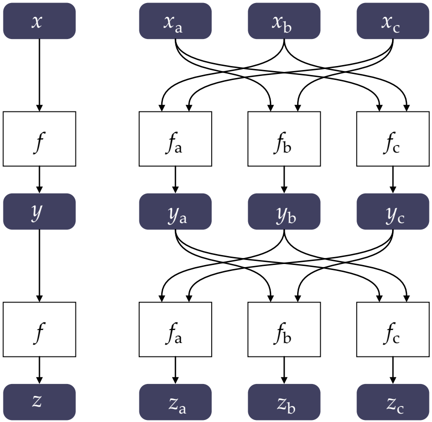
A 3-share threshold implementation



- Uniformly shared variable x:
  - All (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable
  - Equivalently: $\forall x : (x_b, x_c)$ uniform
  - (x_b, x_c): *randomization vector*
- (f_a, f_b, f_c) is sharing of f
- Given x, mapping from (x_b, x_c) to (y_b, y_c) is deterministic:

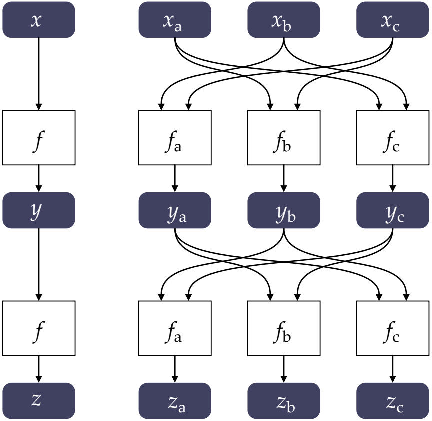
$$(y_b, y_c) = (f_b(x + x_b + x_c, x_c), f_c(x + x_b + x_c, x_b))$$

Relevance of uniformity of a sharing



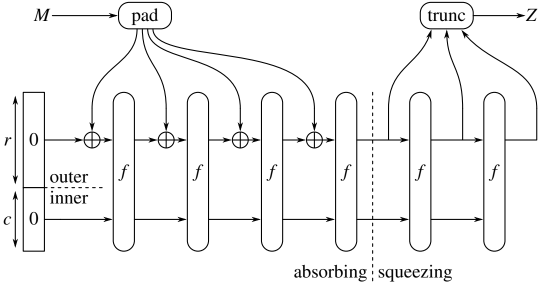
- Sharing (f_a, f_b, f_c) is uniform if
  - $\forall x$: uniform (x_b, x_c) implies uniform (y_b, y_c)
  - if (f_a, f_b, f_c) is a permutation, the sharing is uniform

Relevance of uniformity of a sharing



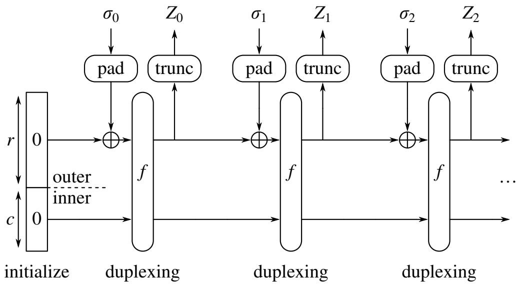
- Sharing (f_a, f_b, f_c) is uniform if
  - $\forall x$: uniform (x_b, x_c) implies uniform (y_b, y_c)
  - if (f_a, f_b, f_c) is a permutation, the sharing is uniform

KECCAK structure: sponge and duplex



- *f*: iterative permutation
- Keyed mode: part of input is secret key
- Security relies on secrecy of inner state
- Try extracting it with side-channel attacks

KECCAK structure: sponge and duplex



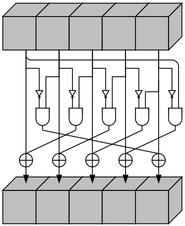
- f: iterative permutation
- Keyed mode: part of input is secret key
- Security relies on secrecy of inner state
- Try extracting it with side-channel attacks

The KECCAK-f round function

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

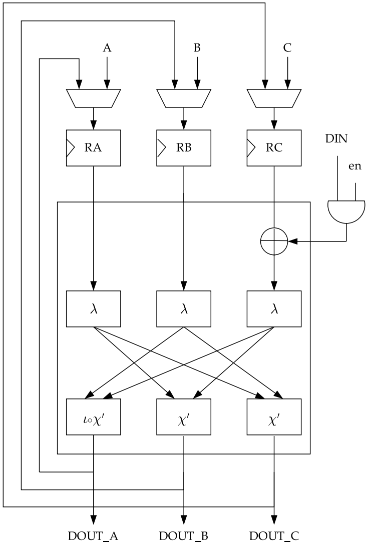
Linear part λ followed by non-linear part χ

- $\lambda = \pi \circ \rho \circ \theta$: mixing followed by bit transposition
- χ: operates on 5-bit *rows*: $y_i = x_i + (x_{i+1} + 1)x_{i+2}$



Example of a 3-share KECCAK architecture

- Randomness initialization:
  - registers a and b: random bits
  - register $c = a + b$
  - once per device power-up
- Resetting the native state to 0
  - reset register $c = 0$
  - fill register with $c = a + b$
  - once per keyed KECCAK instance
- Input absorbed in single share



The sharing of χ

The nonlinear step χ (cyclically on 5-bit rows):

$$X_i \leftarrow \chi_i(x) \triangleq x_i + (x_{i+1} + 1)x_{i+2}$$

Sharing χ' [BDPV SHA-3 2011]:

$$\begin{aligned}
A_i &\leftarrow b_i + (b_{i+1} + 1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1} \\
B_i &\leftarrow c_i + (c_{i+1} + 1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1} \\
C_i &\leftarrow a_i + (a_{i+1} + 1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1}
\end{aligned}$$

χ' is not a permutation and not uniform!

In general, finding efficient uniform threshold sharings is a popular research subject

The sharing of χ

The nonlinear step χ (cyclically on 5-bit rows):

$$X_i \leftarrow \chi_i(x) \triangleq x_i + (x_{i+1} + 1)x_{i+2}$$

Sharing χ' [BDPV SHA-3 2011]:

$$\begin{aligned}
A_i &\leftarrow b_i + (b_{i+1} + 1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1} \\
B_i &\leftarrow c_i + (c_{i+1} + 1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1} \\
C_i &\leftarrow a_i + (a_{i+1} + 1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1}
\end{aligned}$$

χ' is not a permutation and not uniform!

In general, finding efficient uniform threshold sharings is a popular research subject

The sharing of χ

The nonlinear step χ (cyclically on 5-bit rows):

$$X_i \leftarrow \chi_i(x) \triangleq x_i + (x_{i+1} + 1)x_{i+2}$$

Sharing χ' [BDPV SHA-3 2011]:

$$\begin{aligned} A_i &\leftarrow b_i + (b_{i+1} + 1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1} \\ B_i &\leftarrow c_i + (c_{i+1} + 1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1} \\ C_i &\leftarrow a_i + (a_{i+1} + 1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1} \end{aligned}$$

χ' is not a permutation and not uniform!

In general, finding efficient uniform threshold sharings is a popular research subject

Non-uniformity of the χ' sharing

- Two concerns:
  - long-term: randomness in randomization vector evaporates
  - short-term: input to *next* round is not uniform
- First attempt: restore uniformity [BDNNRV Cardis '13]
  - 3-share uniform threshold scheme for χ seems not to exist
  - increase number of shares to 4
  - inject additional randomness: 4 bits per KECCAK-f round
- Second attempt: measure damage [JDA presentations 2015]
  - see what can be done
  - at the expense of giving up on some *provable security*
  - result: cheap trick (almost) removing non-uniformity
- Third attempt [JDA, IACR ePrint (Nov.) 2016/1061]
  - tweaking the trick, effectively achieving uniformity
  - uniformly share any balanced degree-d S-box in $d+1$ shares

Non-uniformity of the χ' sharing

- Two concerns:
  - long-term: randomness in randomization vector evaporates
  - short-term: input to *next* round is not uniform
- First attempt: restore uniformity [BDNNRV Cardis '13]
  - 3-share uniform threshold scheme for χ seems not to exist
  - increase number of shares to 4
  - inject additional randomness: 4 bits per KECCAK-*f* round
- Second attempt: measure damage [JDA presentations 2015]
  - see what can be done
  - at the expense of giving up on some *provable security*
  - result: cheap trick (almost) removing non-uniformity
- Third attempt [JDA, IACR ePrint (Nov.) 2016/1061]
  - tweaking the trick, effectively achieving uniformity
  - uniformly share any balanced degree-d S-box in $d+1$ shares

Non-uniformity of the χ' sharing

- Two concerns:
  - long-term: randomness in randomization vector evaporates
  - short-term: input to *next* round is not uniform
- First attempt: restore uniformity [BDNNRV Cardis '13]
  - 3-share uniform threshold scheme for χ seems not to exist
  - increase number of shares to 4
  - inject additional randomness: 4 bits per KECCAK-*f* round
- Second attempt: measure damage [JDA presentations 2015]
  - see what can be done
  - at the expense of giving up on some *provable security*
  - result: cheap trick (almost) removing non-uniformity
- Third attempt [JDA, IACR ePrint (Nov.) 2016/1061]
  - tweaking the trick, effectively achieving uniformity
  - uniformly share any balanced degree-d S-box in $d+1$ shares

Non-uniformity of the χ' sharing

- Two concerns:
  - long-term: randomness in randomization vector evaporates
  - short-term: input to *next* round is not uniform
- First attempt: restore uniformity [BDNNRV Cardis '13]
  - 3-share uniform threshold scheme for χ seems not to exist
  - increase number of shares to 4
  - inject additional randomness: 4 bits per KECCAK-f round
- Second attempt: measure damage [JDA presentations 2015]
  - see what can be done
  - at the expense of giving up on some *provable security*
  - result: cheap trick (almost) removing non-uniformity
- Third attempt [JDA, IACR ePrint (Nov.) 2016/1061]
  - tweaking the trick, effectively achieving uniformity
  - uniformly share any balanced degree-d S-box in $d+1$ shares

Non-uniformity of the χ' sharing

- Two concerns:
  - long-term: randomness in randomization vector evaporates
  - short-term: input to *next* round is not uniform
- First attempt: restore uniformity [BDNNRV Cardis '13]
  - 3-share uniform threshold scheme for χ seems not to exist
  - increase number of shares to 4
  - inject additional randomness: 4 bits per KECCAK-*f* round
- Second attempt: measure damage [JDA presentations 2015]
  - see what can be done
  - at the expense of giving up on some *provable security*
  - result: cheap trick (almost) removing non-uniformity
- Third attempt [JDA, IACR ePrint (Nov.) 2016/1061]
  - tweaking the trick, effectively achieving uniformity
  - uniformly share any balanced degree-d S-box in $d+1$ shares

Outline

1 The original motivation

2 Distributions, spectrum and collision probability

3 Mappings and correlation matrices

4 In the setting of a shared implementation

5 Achieving uniformity

Imbalance spectrum of a distribution

- n-bit variable x with a given distribution $X(x)$
- Imbalance of a bit of x: indicates probability that it is 0 or 1
- Imbalance of a parity of x, defined by mask v:

$$\widetilde{X}[v] = \sum_x X(x)(-1)^{v^\top \times x}$$

- Vector of imbalances for all 2^n masks v: imbalance spectrum $\widetilde{X}$

Imbalance spectrum of a distribution

- Vector of imbalances for all 2^n masks v: imbalance spectrum $\widetilde{X}$
  - Note: $\forall X, \widetilde{X}[0] = 1$
  - Reduced spectrum $\widehat{X}$: $\widetilde{X}$ with $\widetilde{X}[0]$ removed
- Total imbalance: energy in $\widehat{X}$:

$$\phi_X = ||\widehat{X}||^2 = \sum_{v \neq 0} (\widetilde{X}[v])^2$$

  - $\phi_X = 0$: uniform distribution, entropy n bits
  - $\phi_X = 2^n - 1$: peak distribution, entropy 0 bits

Imbalance spectrum of a distribution

- Vector of imbalances for all 2^n masks v: imbalance spectrum $\widetilde{X}$
  - Note: $\forall X, \widetilde{X}[0] = 1$
  - Reduced spectrum $\widehat{X}$: $\widetilde{X}$ with $\widetilde{X}[0]$ removed
- Total imbalance: energy in $\widehat{X}$:

$$\phi_X = ||\widehat{X}||^2 = \sum_{v \neq 0} (\widetilde{X}[v])^2$$

  - $\phi_X = 0$: uniform distribution, entropy n bits
  - $\phi_X = 2^n - 1$: peak distribution, entropy 0 bits

Imbalance spectrum of a distribution

- Vector of imbalances for all 2^n masks v: imbalance spectrum $\widetilde{X}$
  - Note: $\forall X, \widetilde{X}[0] = 1$
  - Reduced spectrum $\widehat{X}$: $\widetilde{X}$ with $\widetilde{X}[0]$ removed
- Total imbalance: energy in $\widehat{X}$:

$$\phi_X = ||\widehat{X}||^2 = \sum_{v \neq 0} (\widetilde{X}[v])^2$$

  - $\phi_X = 0$: uniform distribution, entropy n bits
  - $\phi_X = 2^n - 1$: peak distribution, entropy 0 bits

Outline

1 The original motivation

2 Distributions, spectrum and collision probability

3 Mappings and correlation matrices

4 In the setting of a shared implementation

5 Achieving uniformity

Correlation matrices

- Mapping from m to n bits: $f(x) = (f_1(x), f_2(x) \ldots f_n(x))$
- Correlation matrix $C^{(f)}$:
  - 2^n rows and 2^m columns
  - element at row u, column v: $C(u^{\mathsf{T}} \times f(x), v^{\mathsf{T}} \times x)$
- Homomorphism:

$$\begin{tikzcd}
x \arrow[r, "f"] \arrow[d, Leftrightarrow, "\mathcal{L}"] & y = f(x) \arrow[d, Leftrightarrow, "\mathcal{L}"] \\
\alpha \text{ with } \alpha_u = (-1)^{x^{\mathsf{T}} \times u} \arrow[r, "C^{(f)}"] & \beta = C^{(f)} \times \alpha \text{ with } \beta_u = (-1)^{y^{\mathsf{T}} \times u}
\end{tikzcd}$$

- If f is an n-bit permutation: $C^{(f^{-1})} = \left(C^{(f)}\right)^{-1} = \left(C^{(f)}\right)^{\mathsf{T}}$

Correlation matrices

- Mapping from m to n bits: $f(x) = (f_1(x), f_2(x) \dots f_n(x))$
- Correlation matrix $C^{(f)}$:
  - 2^n rows and 2^m columns
  - element at row u, column v: $C(u^\mathsf{T} \times f(x), v^\mathsf{T} \times x)$
- Homomorphism:

$$\begin{array}{ccc}
x & \xrightarrow{\quad f \quad} & y = f(x) \\
\Updownarrow \mathcal{L} & & \Updownarrow \mathcal{L} \\
\alpha \text{ with } \alpha_u = (-1)^{x^\mathsf{T} \times u} & \xrightarrow{\quad C^{(f)} \quad} & \beta = C^{(f)} \times \alpha \text{ with } \beta_u = (-1)^{y^\mathsf{T} \times u}
\end{array}$$

- If f is an n-bit permutation: $C^{(f^{-1})} = \left(C^{(f)}\right)^{-1} = \left(C^{(f)}\right)^\mathsf{T}$

Imbalance spectrum propagation

Let $y = f(x)$, then

$$\widetilde{Y} = C^{(f)} \times \widetilde{X}$$

With reduced spectra:

$$\begin{bmatrix} 1 \\ \widehat{Y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ I^f & C^{*(f)} \end{bmatrix} \times \begin{bmatrix} 1 \\ \widehat{X} \end{bmatrix}$$

So:

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

- I^f imbalance vector of f
- $\phi_f = ||I^f||^2$: imbalance contribution of f

Imbalance spectrum propagation

Let $y = f(x)$, then

$$\widetilde{Y} = C^{(f)} \times \widetilde{X}$$

With reduced spectra:

$$\begin{bmatrix} 1 \\ \widehat{Y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ I^f & C^{*(f)} \end{bmatrix} \times \begin{bmatrix} 1 \\ \widehat{X} \end{bmatrix}$$

So:

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

- I^f imbalance vector of f
- $\phi_f = ||I^f||^2$: imbalance contribution of f

Imbalance spectrum propagation

Let $y = f(x)$, then

$$\widetilde{Y} = C^{(f)} \times \widetilde{X}$$

With reduced spectra:

$$\begin{bmatrix} 1 \\ \widehat{Y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ I^f & C^{*(f)} \end{bmatrix} \times \begin{bmatrix} 1 \\ \widehat{X} \end{bmatrix}$$

So:

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

- I^f imbalance vector of f
- $\phi_f = ||I^f||^2$: imbalance contribution of f

Imbalance spectrum propagation

Let $y = f(x)$, then

$$\widetilde{Y} = C^{(f)} \times \widetilde{X}$$

With reduced spectra:

$$\begin{bmatrix} 1 \\ \widehat{Y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ I^f & C^{*(f)} \end{bmatrix} \times \begin{bmatrix} 1 \\ \widehat{X} \end{bmatrix}$$

So:

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

- I^f imbalance vector of f
- $\phi_f = ||I^f||^2$: imbalance contribution of f

Macroscopic perspective: total imbalances

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

Assuming orthogonality:

$$||\widehat{Y}||^2 \approx ||I^f||^2 + ||C^{*(f)} \times \widehat{X}||^2$$

or (assuming $\phi_f \lll 2^n$)

$$\phi_Y \approx \phi_X + \phi_f$$

For $y = f(x)$ with $f = f_r \circ f_{r-1} \cdots \circ f_1$ this gives:

$$\phi_Y \approx \phi_X + \sum_i \phi_{f_i}$$

Total imbalance increases linearly with number of rounds

Macroscopic perspective: total imbalances

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

Assuming orthogonality:

$$||\widehat{Y}||^2 \approx ||I^f||^2 + ||C^{*(f)} \times \widehat{X}||^2$$

or (assuming $\phi_f \lll 2^n$)

$$\phi_Y \approx \phi_X + \phi_f$$

For $y = f(x)$ with $f = f_r \circ f_{r-1} \cdots \circ f_1$ this gives:

$$\phi_Y \approx \phi_X + \sum_i \phi_{f_i}$$

Total imbalance increases linearly with number of rounds

Macroscopic perspective: total imbalances

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

Assuming orthogonality:

$$||\widehat{Y}||^2 \approx ||I^f||^2 + ||C^{*(f)} \times \widehat{X}||^2$$

or (assuming $\phi_f \lll 2^n$)

$$\phi_Y \approx \phi_X + \phi_f$$

For $y = f(x)$ with $f = f_r \circ f_{r-1} \cdots \circ f_1$ this gives:

$$\phi_Y \approx \phi_X + \sum_i \phi_{f_i}$$

Total imbalance increases linearly with number of rounds

Macroscopic perspective: total imbalances

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

Assuming orthogonality:

$$||\widehat{Y}||^2 \approx ||I^f||^2 + ||C^{*(f)} \times \widehat{X}||^2$$

or (assuming $\phi_f \lll 2^n$)

$$\phi_Y \approx \phi_X + \phi_f$$

For $y = f(x)$ with $f = f_r \circ f_{r-1} \cdots \circ f_1$ this gives:

$$\phi_Y \approx \phi_X + \sum_i \phi_{f_i}$$

Total imbalance increases linearly with number of rounds

Macroscopic perspective: total imbalances

$$\widehat{Y} = I^f + C^{*(f)} \times \widehat{X}$$

Assuming orthogonality:

$$||\widehat{Y}||^2 \approx ||I^f||^2 + ||C^{*(f)} \times \widehat{X}||^2$$

or (assuming $\phi_f \lll 2^n$)

$$\phi_Y \approx \phi_X + \phi_f$$

For $y = f(x)$ with $f = f_r \circ f_{r-1} \cdots \circ f_1$ this gives:

$$\phi_Y \approx \phi_X + \sum_i \phi_{f_i}$$

Total imbalance increases linearly with number of rounds

Microscopic perspective: individual imbalances

Single round f (assuming uniform input):

$$\widetilde{Y}[v] = I^{f}[v]$$

Two rounds $f_0 \circ f_{-1}$

$$\widehat{Y} = I^{f_0} + C^{*(f_0)} \times I^{f_{-1}}$$

So

$$\widetilde{Y}[v] = I^{f_0}[v] + \sum_{w} C^{*(f_0)}_{vw} \times I^{f_{-1}}[w]$$

Elements in $I^{f_{-1}}[w]$ are multiplied by elements of correlation matrix

Microscopic perspective: individual imbalances

Single round f (assuming uniform input):

$$\widetilde{Y}[v] = I^{f}[v]$$

Two rounds $f_0 \circ f_{-1}$

$$\widehat{Y} = I^{f_0} + C^{*(f_0)} \times I^{f_{-1}}$$

So

$$\widetilde{Y}[v] = I^{f_0}[v] + \sum_{w} C^{*(f_0)}_{vw} \times I^{f_{-1}}[w]$$

Elements in $I^{f_{-1}}[w]$ are multiplied by elements of correlation matrix

Microscopic perspective: individual imbalances

Single round f (assuming uniform input):

$$\widetilde{Y}[v] = I^{f}[v]$$

Two rounds $f_0 \circ f_{-1}$

$$\widehat{Y} = I^{f_0} + C^{*(f_0)} \times I^{f_{-1}}$$

So

$$\widetilde{Y}[v] = I^{f_0}[v] + \sum_{w} C^{*(f_0)}_{vw} \times I^{f_{-1}}[w]$$

Elements in $I^{f_{-1}}[w]$ are multiplied by elements of correlation matrix

Microscopic perspective: individual imbalances

Single round f (assuming uniform input):

$$\widetilde{Y}[v] = I^{f}[v]$$

Two rounds $f_0 \circ f_{-1}$

$$\widehat{Y} = I^{f_0} + C^{*(f_0)} \times I^{f_{-1}}$$

So

$$\widetilde{Y}[v] = I^{f_0}[v] + \sum_{w} C^{*(f_0)}_{vw} \times I^{f_{-1}}[w]$$

Elements in $I^{f_{-1}}[w]$ are multiplied by elements of correlation matrix

Microscopic perspective (cont'd)

Adding a round f_{-r}

$$\widehat{Y} = \ldots + \prod_{0 \le i < r} C^{*(f_{-i})} \times I^{f_{-r}}$$

In terms of linear trails Q

$$\widetilde{Y}[v] = \ldots + \sum_{w} \left(\sum_{Q \text{ with } q_{-r}=w \text{ and } q_0=v} C_Q \right) I^{f_{-r}}[w]$$

- Imbalances in $I^{f_{-r}}$ multiplied by trail correlation contributions C_Q
- Energy from $I^{f_{-r}}$ more and more diffused as r increases

Microscopic perspective (cont'd)

Adding a round f_{-r}

$$\widehat{Y} = \ldots + \prod_{0 \leq i < r} C^{*(f_{-i})} \times I^{f_{-r}}$$

In terms of linear trails Q

$$\widetilde{Y}[v] = \ldots + \sum_{w} \left(\sum_{Q \text{ with } q_{-r}=w \text{ and } q_0=v} C_Q \right) I^{f_{-r}}[w]$$

- Imbalances in $I^{f_{-r}}$ multiplied by trail correlation contributions C_Q
- Energy from $I^{f_{-r}}$ more and more diffused as r increases

Microscopic perspective (cont'd)

Adding a round f_{-r}

$$\widehat{Y} = \ldots + \prod_{0 \leq i < r} C^{*(f_{-i})} \times I^{f_{-r}}$$

In terms of linear trails Q

$$\widetilde{Y}[v] = \ldots + \sum_{w} \left(\sum_{Q \text{ with } q_{-r}=w \text{ and } q_0=v} C_Q \right) I^{f_{-r}}[w]$$

- Imbalances in $I^{f_{-r}}$ multiplied by trail correlation contributions C_Q
- Energy from $I^{f_{-r}}$ more and more diffused as r increases

Microscopic perspective (cont'd)

Adding a round f_{-r}

$$\widehat{Y} = \ldots + \prod_{0 \leq i < r} C^{*(f_{-i})} \times I^{f_{-r}}$$

In terms of linear trails Q

$$\widetilde{Y}[v] = \ldots + \sum_{w} \left(\sum_{Q \text{ with } q_{-r}=w \text{ and } q_0=v} C_Q \right) I^{f_{-r}}[w]$$

- Imbalances in $I^{f_{-r}}$ multiplied by trail correlation contributions C_Q
- Energy from $I^{f_{-r}}$ more and more diffused as r increases

Microscopic perspective (cont'd)

Adding a round f_{-r}

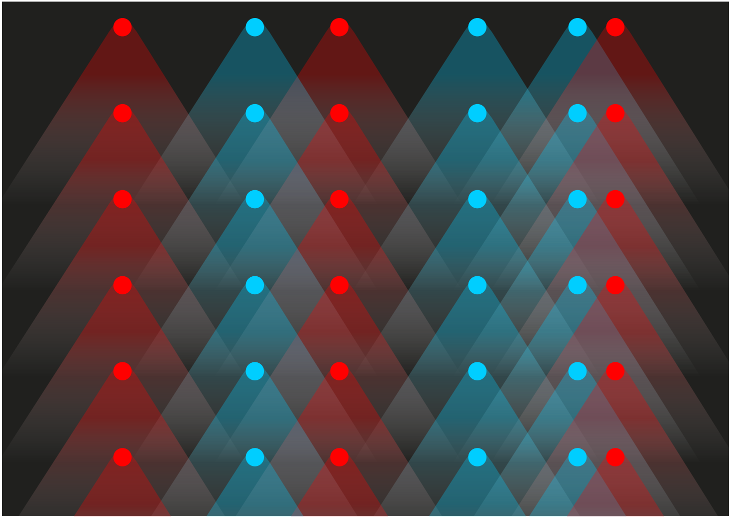
$$\widehat{Y} = \ldots + \prod_{0 \leq i < r} C^{*(f_{-i})} \times I^{f_{-r}}$$

In terms of linear trails Q

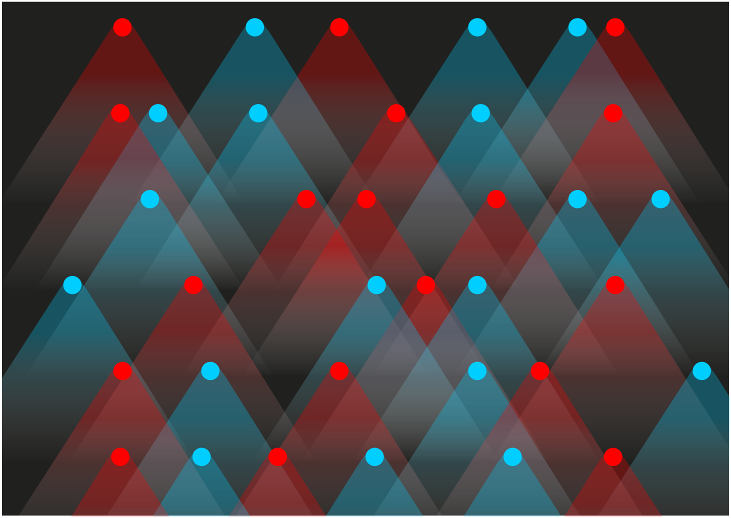
$$\widetilde{Y}[v] = \ldots + \sum_{w} \left(\sum_{Q \text{ with } q_{-r}=w \text{ and } q_0=v} C_Q \right) I^{f_{-r}}[w]$$

- Imbalances in $I^{f_{-r}}$ multiplied by trail correlation contributions C_Q
- Energy from $I^{f_{-r}}$ more and more diffused as r increases

Visualization: iteration of a single transformation f



Visualization: composition of transformations $f_{(i)}$



Outline

1. The original motivation
2. Distributions, spectrum and collision probability
3. Mappings and correlation matrices
4. In the setting of a shared implementation
5. Achieving uniformity

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
    - all have same native state sequence x
    - partially unknown
    - different inputs at the end for doing DPA
    - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Our DPA setting

- Shared implementation
- Data complexity: z traces
  - all have same native state sequence x
  - partially unknown
  - different inputs at the end for doing DPA
  - initial state: z independent randomization vectors x_b, x_c
- Operations recorded in a trace consists of iterations of round function interleaved with round constant addition or absorbing input
- We study the propagation of imbalances in the randomization vector

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share Keccak implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Analyzing propagation

- Determine relevant masks (v_b, v_c)
  - A combinatorial circuit may leak if its input is non-uniform
  - Identify the *smallest* combinatorial blocks
  - Trace each output bit to its inputs: *support range*
  - Don't forget to include multiplexers etc.
  - Specific for the hardware architecture
  - Relevant masks (v_b, v_c): those that are zero outside support range
- Investigate propagation of imbalance vectors $I^{R'[x]}$
  - of previous round
  - of round before
  - ...as many rounds as considered relevant
- Preliminary conclusion for three-share KECCAK implementation:
  - exploiting non-uniformity likely harder than higher-order attacks
  - but it would be nice to get rid of non-uniformity nonetheless

Outline

1 The original motivation

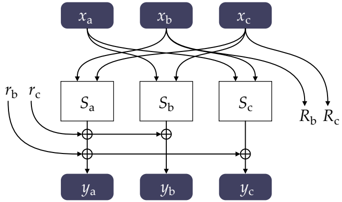
2 Distributions, spectrum and collision probability

3 Mappings and correlation matrices

4 In the setting of a shared implementation

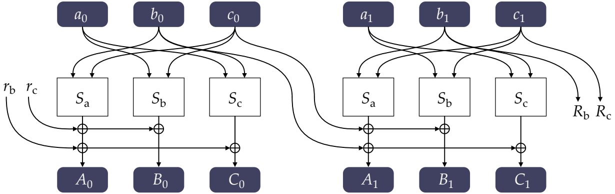
5 Achieving uniformity

Borrowing randomness from the neighbours



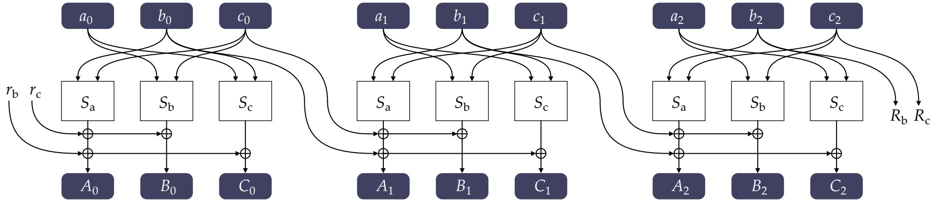
- Repair uniformity by taking randomness from neighbour's input
- Does not affect correctness and incompleteness
- Works for 3 shares, generalizes to any number of shares
- Output is uniformly shared if
  - Input is uniformly shared
  - (r_b, r_c) has a uniform distribution

Borrowing randomness from the neighbours



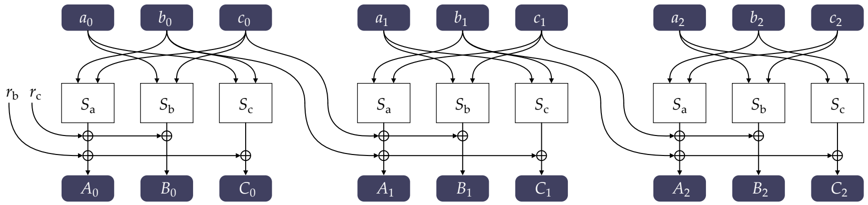
- Repair uniformity by taking randomness from neighbour's input
- Does not affect correctness and incompleteness
- Works for 3 shares, generalizes to any number of shares
- Output is uniformly shared if
  - Input is uniformly shared
  - (r_b, r_c) has a uniform distribution

Borrowing randomness from the neighbours



- Repair uniformity by taking randomness from neighbour's input
- Does not affect correctness and incompleteness
- Works for 3 shares, generalizes to any number of shares
- Output is uniformly shared if
  - Input is uniformly shared
  - (r_b, r_c) has a uniform distribution

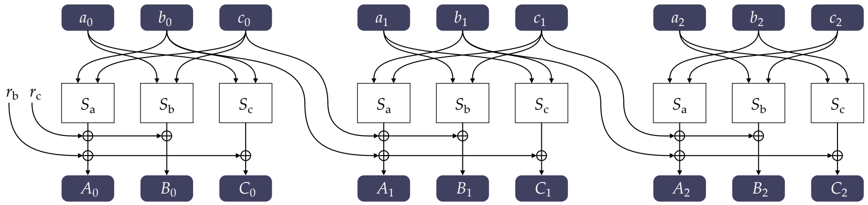
Solution 1: injecting randomness



- (r_b, r_c) are generated freshly every round
- (R_b, R_c) thrown away
- If S-box width is n bits, requires $2n$ random bits per round
- KECCAK: $n = 5$, but we can reduce to 4 random bits

Was already presented and proven secure in [Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

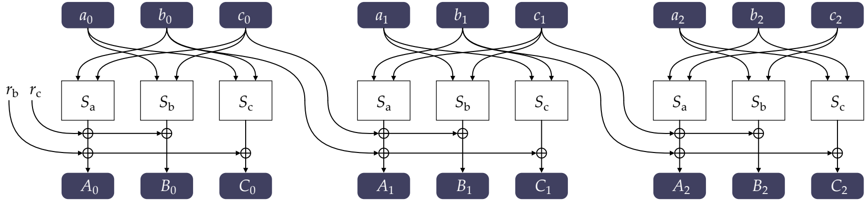
Solution 1: injecting randomness



- (r_b, r_c) are generated freshly every round
- (R_b, R_c) thrown away
- If S-box width is n bits, requires $2n$ random bits per round
- KECCAK: $n = 5$, but we can reduce to 4 random bits

Was already presented and proven secure in [Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

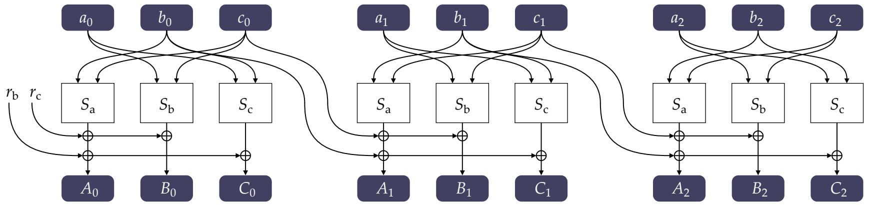
Solution 1: injecting randomness



- (r_b, r_c) are generated freshly every round
- (R_b, R_c) thrown away
- If S-box width is n bits, requires $2n$ random bits per round
- KECCAK: $n = 5$, but we can reduce to 4 random bits

Was already presented and proven secure in [Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

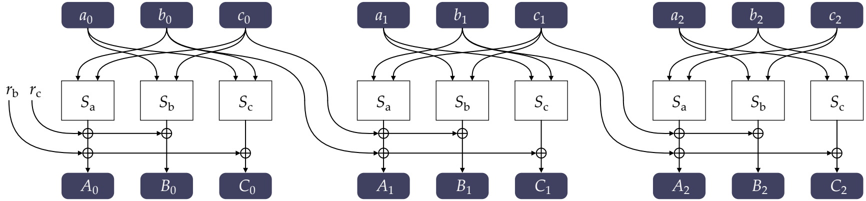
Solution 1: injecting randomness



- (r_b, r_c) are generated freshly every round
- (R_b, R_c) thrown away
- If S-box width is n bits, requires $2n$ random bits per round
- KECCAK: $n = 5$, but we can reduce to 4 random bits

Was already presented and proven secure in [Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

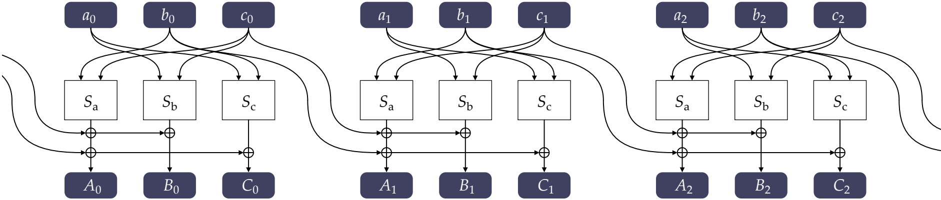
Solution 1: injecting randomness



- (r_b, r_c) are generated freshly every round
- (R_b, R_c) thrown away
- If S-box width is n bits, requires $2n$ random bits per round
- KECCAK: $n = 5$, but we can reduce to 4 random bits

Was already presented and proven secure in [Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

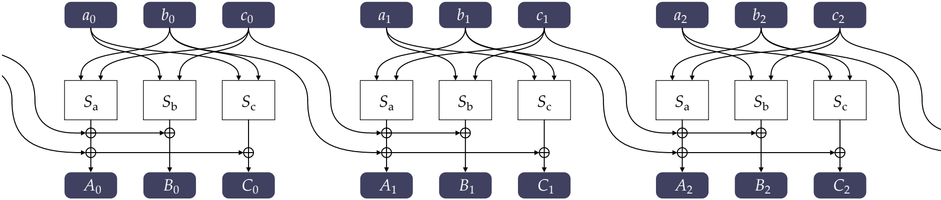
Solution 2: cycling randomness



- $(r_b, r_c) = (R_b, R_c)$
- S-boxes are arranged in circle
- No more need for generating randomness
- Some non-uniformity remains ...

Presented earlier at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

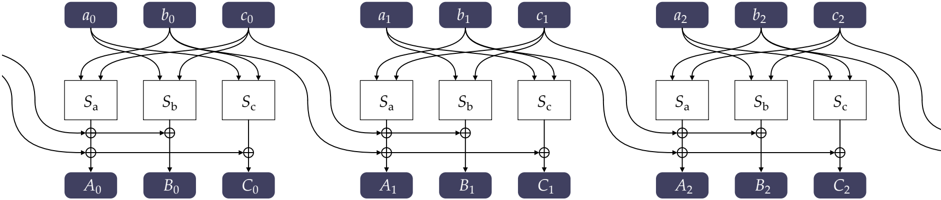
Solution 2: cycling randomness



- $(r_b, r_c) = (R_b, R_c)$
- S-boxes are arranged in circle
- No more need for generating randomness
- Some non-uniformity remains ...

Presented earlier at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

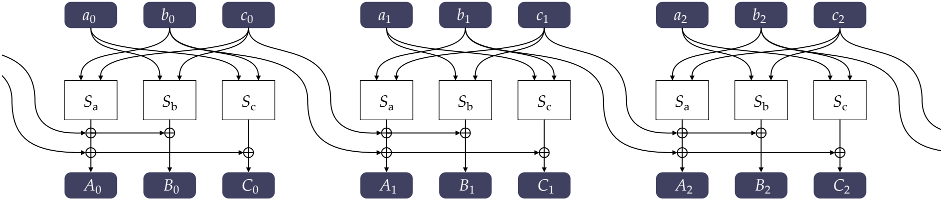
Solution 2: cycling randomness



- $(r_b, r_c) = (R_b, R_c)$
- S-boxes are arranged in circle
- No more need for generating randomness
- Some non-uniformity remains ...

Presented earlier at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

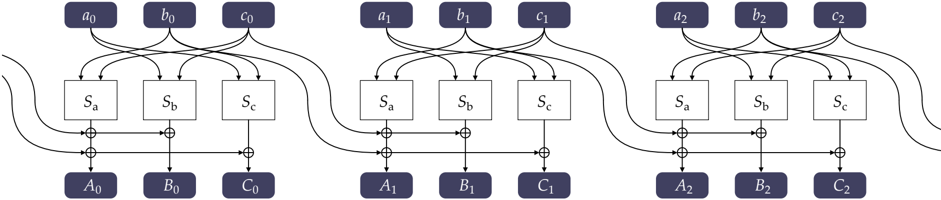
Solution 2: cycling randomness



- $(r_b, r_c) = (R_b, R_c)$
- S-boxes are arranged in circle
- No more need for generating randomness
- Some non-uniformity remains ...

Presented earlier at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

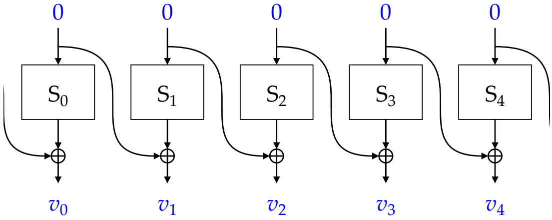
Solution 2: cycling randomness



- $(r_b, r_c) = (R_b, R_c)$
- S-boxes are arranged in circle
- No more need for generating randomness
- Some non-uniformity remains ...

Presented earlier at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

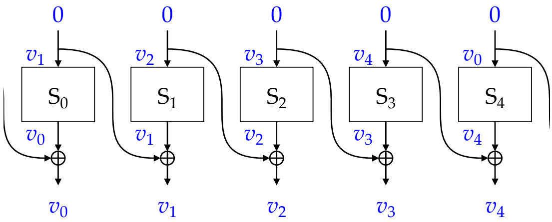
Conceptual mapping of (x_b, x_c) to (y_b, y_c)



Properties of imbalance vector I^β

- $C(0, u) = 0$ if $u \neq 0$, for any possible S-box
- $C^\alpha(v, 0) \neq 0 \Rightarrow \forall i : v_i = 0$
- Non-zero elements of imbalance vector I^β
  - are active in all S-boxes
  - amplitude $\leq \left(\max_{(u,v)} C^{(S_{a,b,c})}(u, v)\right)^{\#\text{S-boxes}}$
  - e.g., $\leq 2^{-80}$ for KECCAK-$f[200]$

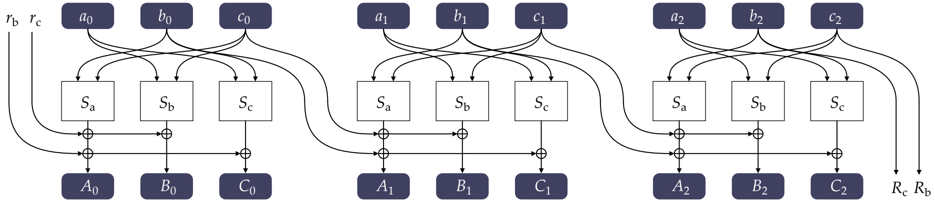
Conceptual mapping of (x_b, x_c) to (y_b, y_c)



Properties of imbalance vector I^β

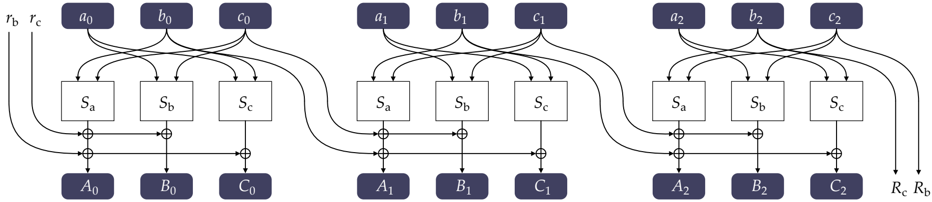
- $C(0, u) = 0$ if $u \neq 0$, for any possible S-box
- $C^\alpha(v, 0) \neq 0 \Rightarrow \forall i : v_i = 0$
- Non-zero elements of imbalance vector I^β
  - are active in all S-boxes
  - amplitude $\leq (\max_{(u,v)} C^{(S_{a,b,c})}(u, v))^{\#\text{S-boxes}}$
  - e.g., $\leq 2^{-80}$ for KECCAK-$f[200]$

Solution 3: recycling randomness (New!)



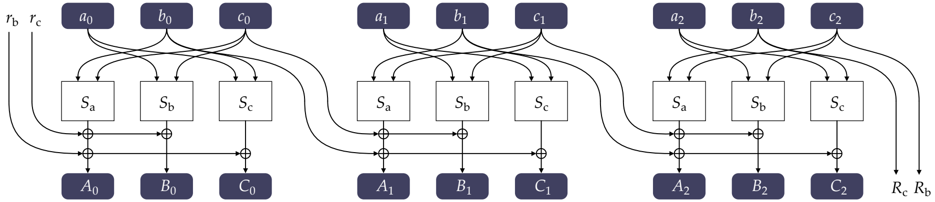
- (R_b, R_c) is part of the shared state
- (r_b, r_c) is (R_b, R_c) from previous round
- Achieves uniformity if S-box is invertible
- Cost:
  - 4 additional XORs per native bit
  - shared state extended by $2n$ additional bits (for n-bit S-box)

Solution 3: recycling randomness (New!)



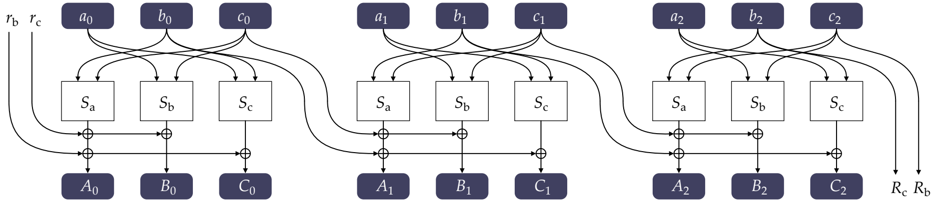
- (R_b, R_c) is part of the shared state
- (r_b, r_c) is (R_b, R_c) from previous round
- Achieves uniformity if S-box is invertible
- Cost:
  - 4 additional XORs per native bit
  - shared state extended by $2n$ additional bits (for n-bit S-box)

Solution 3: recycling randomness (New!)



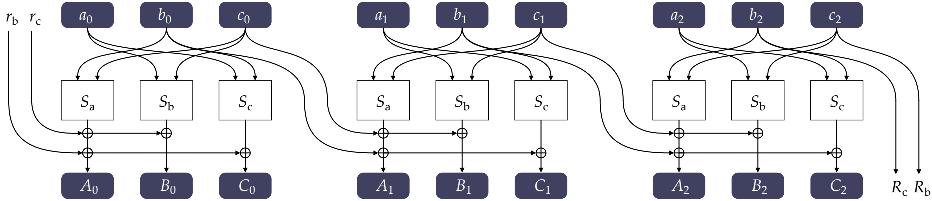
- (R_b, R_c) is part of the shared state
- (r_b, r_c) is (R_b, R_c) from previous round
- Achieves uniformity if S-box is invertible
- Cost:
  - 4 additional XORs per native bit
  - shared state extended by $2n$ additional bits (for n-bit S-box)

Solution 3: recycling randomness (New!)



- (R_b, R_c) is part of the shared state
- (r_b, r_c) is (R_b, R_c) from previous round
- Achieves uniformity if S-box is invertible
- Cost:
  - 4 additional XORs per native bit
  - shared state extended by $2n$ additional bits (for n-bit S-box)

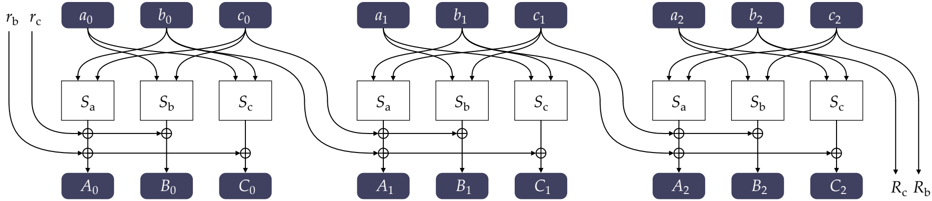
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

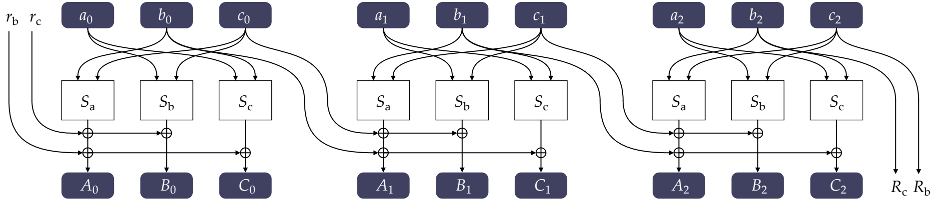
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

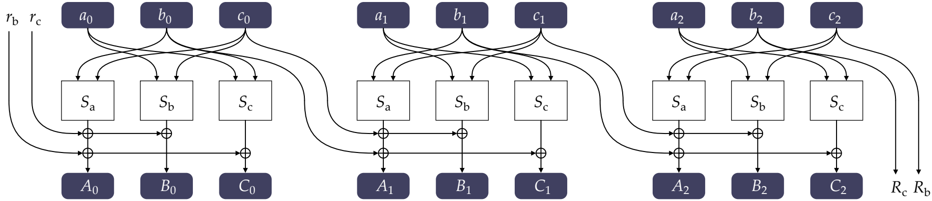
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

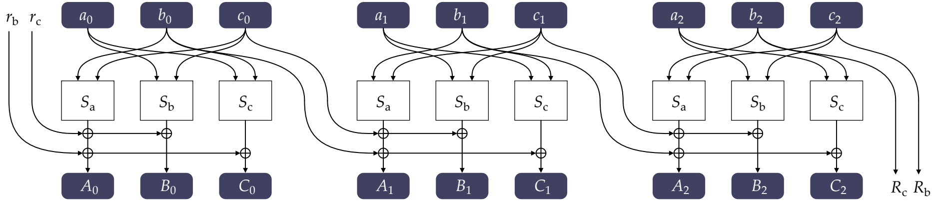
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

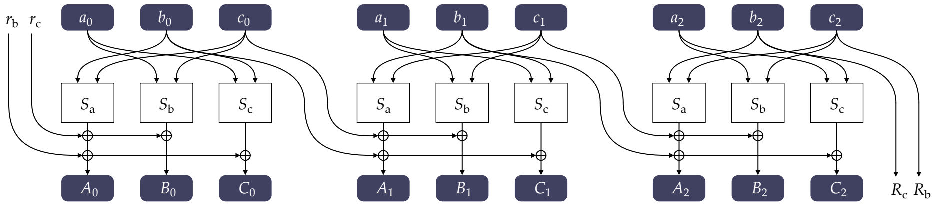
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

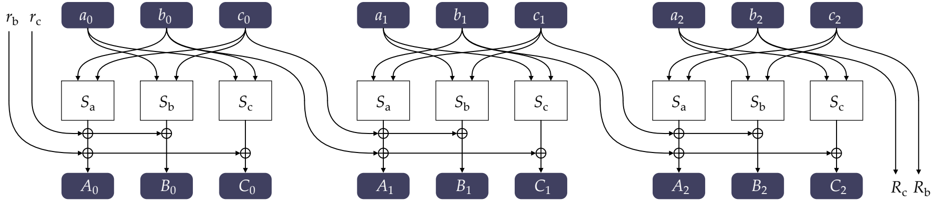
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

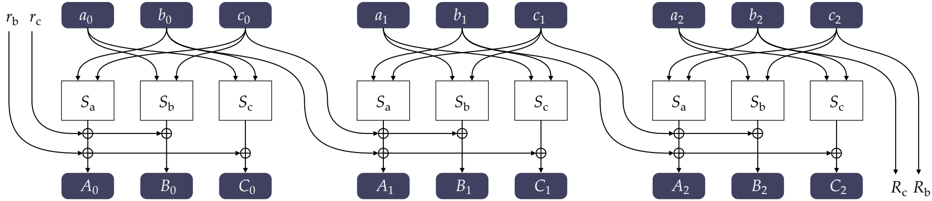
Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

Solution 3: proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

Application to χ'

$$
\begin{aligned}
A_i &\leftarrow b_i + (b_{i+1}+1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1}\\
B_i &\leftarrow c_i + (c_{i+1}+1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1}\\
C_i &\leftarrow a_i + (a_{i+1}+1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1}
\end{aligned}
$$

- *Multipermutation property* of χ':
  - let $x|_\ell = (x_0, x_1)$: left part of x
  - let $x|_r = (x_2, x_3, x_4)$: right part of x
  - Mapping from $((a,b,c)|_\ell, (A,B,C)|_r)$ to $((a,b,c)|_r, (A,B,C)|_\ell)$ is permutation
- This allows us to
  - only add randomness to left part: 8 XOR gates per row
  - limit r_b and r_c each to two bits

Application to χ'

$$A_i \leftarrow b_i + (b_{i+1}+1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1}$$
$$B_i \leftarrow c_i + (c_{i+1}+1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1}$$
$$C_i \leftarrow a_i + (a_{i+1}+1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1}$$

- *Multipermutation property* of χ':
  - let $x|_\ell = (x_0, x_1)$: left part of x
  - let $x|_r = (x_2, x_3, x_4)$: right part of x
  - Mapping from $((a,b,c)|_\ell, (A,B,C)|_r)$ to $((a,b,c)|_r, (A,B,C)|_\ell)$ is permutation
- This allows us to
  - only add randomness to left part: 8 XOR gates per row
  - limit r_b and r_c each to two bits

Application to χ'

$$
\begin{aligned}
A_i &\leftarrow b_i + (b_{i+1} + 1)b_{i+2} + b_{i+1}c_{i+2} + b_{i+2}c_{i+1} \\
B_i &\leftarrow c_i + (c_{i+1} + 1)c_{i+2} + c_{i+1}a_{i+2} + c_{i+2}a_{i+1} \\
C_i &\leftarrow a_i + (a_{i+1} + 1)a_{i+2} + a_{i+1}b_{i+2} + a_{i+2}b_{i+1}
\end{aligned}
$$

- *Multipermutation property* of χ':
  - let $x|_\ell = (x_0, x_1)$: left part of x
  - let $x|_r = (x_2, x_3, x_4)$: right part of x
  - Mapping from $((a, b, c)|_\ell, (A, B, C)|_r)$ to $((a, b, c)|_r, (A, B, C)|_\ell)$ is permutation
- This allows us to
  - only add randomness to left part: 8 XOR gates per row
  - limit r_b and r_c each to two bits

Generalization for invertible *n*-bit S-box of degree *d*

- Correct and incomplete sharing: $d+1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i-1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Generalization for invertible n-bit S-box of degree d

- Correct and incomplete sharing: $d + 1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i - 1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Generalization for invertible n-bit S-box of degree d

- Correct and incomplete sharing: $d+1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i-1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Generalization for invertible *n*-bit S-box of degree *d*

- Correct and incomplete sharing: $d + 1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i - 1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Generalization for invertible n-bit S-box of degree d

- Correct and incomplete sharing: $d + 1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i - 1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Generalization for invertible n-bit S-box of degree d

- Correct and incomplete sharing: $d + 1$ shares
- Randomness borrowing:
  - randomization vector R: last d shares
  - each share of R of S-box $i - 1$ added to 2 shares of S-box i
- Total cost due to randomness borrowing (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-permutation property

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?

Conclusions

- Spectral perspective to lossy mappings
- Natural application of correlation matrices
- Technique for achieving provable first-order DPA resistance
  - KECCAK: χ', borrowing at cost 8 XORs per row
  - relatively cheap for any invertible S-box
- Abandon quest for uniformly shareable S-boxes
- Look for low-degree S-boxes with multi-permutation sharing instead

Thanks for your attention!

Q?