Optimising Linear Key Recovery Attacks with Affine Walsh Transform Pruning

Antonio Flórez-Gutiérrez

NTT Social Informatics Laboratories
(work carried out while at Inria Paris)

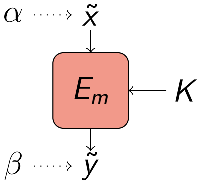
ASIACRYPT 2022
5-9 December 2022, Taipei

Introduction and Motivation

Linear Key Recovery Attack

Linear approximation of (part of) a block cipher (Matsui,1993):

$$\langle \alpha, \tilde{x} \rangle \oplus \langle \beta, \tilde{y} \rangle \text{ with correlation } c$$



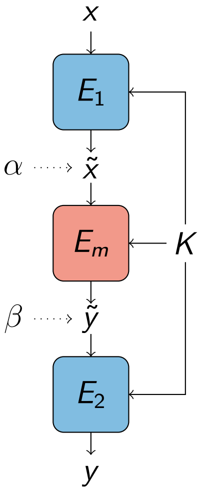
Linear Key Recovery Attack

Linear approximation of (part of) a block cipher (Matsui,1993):

$$\langle \alpha, \tilde{x} \rangle \oplus \langle \beta, \tilde{y} \rangle \text{ with correlation } c$$

We express the linear approximation as a function of the plaintext, ciphertext and key with the key recovery map, for example:

$$f_0(x, y) \oplus f_1(x, K) \oplus f_2(y, K)$$



Linear Key Recovery Attack

Linear approximation of (part of) a block cipher (Matsui,1993):

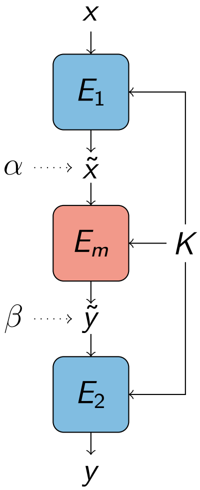
$$\langle \alpha, \tilde{x} \rangle \oplus \langle \beta, \tilde{y} \rangle \text{ with correlation } c$$

We express the linear approximation as a function of the plaintext, ciphertext and key with the key recovery map, for example:

$$f_0(x, y) \oplus f_1(x, K) \oplus f_2(y, K)$$

We divide the relevant part of the plaintext/ciphertext into segments and consider key recovery maps of the form:

$$f_0(x) \oplus \underbrace{f_1(x_1 \oplus k_1^O, k_1^I) \oplus \ldots \oplus f_d(x_d \oplus k_d^O, k_d^I)}_{f(X \oplus K^O, K^I)}$$



Linear Key Recovery Attack *(cont.)*

The objective is to compute all the experimental correlations:

$$\widehat{\mathrm{cor}}(\text{key guess } k) = \frac{1}{N} \sum_{x \in \mathcal{D}} (-1)^{\langle \alpha, \tilde{x}(k) \rangle \oplus \langle \beta, \tilde{y}(k) \rangle}, \; \mathcal{D} \text{ data sample of size } N \approx 1/c^2$$

as the correct key guess is expected to have a larger experimental correlation

Linear Key Recovery Attack *(cont.)*

The objective is to compute all the experimental correlations:

$$\widehat{\mathrm{cor}}(\text{key guess } k) = \frac{1}{N}\sum_{x\in\mathcal{D}}(-1)^{\langle\alpha,\tilde{x}(k)\rangle\oplus\langle\beta,\tilde{y}(k)\rangle},\ \mathcal{D} \text{ data sample of size } N \approx 1/c^2$$

as the correct key guess is expected to have a larger experimental correlation
For the considered form of the key recovery map, we have

$$\widehat{cor}(K^O, K^I) = \frac{1}{N}\sum_{x\in\mathcal{D}}(-1)^{f_0(x)}(-1)^{f(X\oplus K^O, K^I)}$$

Linear Key Recovery Attack *(cont.)*

The objective is to compute all the experimental correlations:

$$\widehat{\mathrm{cor}}(\text{key guess } k) = \frac{1}{N}\sum_{x\in\mathcal{D}}(-1)^{\langle\alpha,\tilde{x}(k)\rangle\oplus\langle\beta,\tilde{y}(k)\rangle},\ \mathcal{D} \text{ data sample of size } N \approx 1/c^2$$

as the correct key guess is expected to have a larger experimental correlation
For the considered form of the key recovery map, we have

$$\widehat{cor}(K^O, K^I) = \frac{1}{N}\sum_{x\in\mathcal{D}}(-1)^{f_0(x)}(-1)^{f(X\oplus K^O, K^I)}$$

We can compute this vector either directly or with a distillation step, with costs

$N \cdot 2^{|K^I|+|K^O|}$ (Matsui, 1993) and $N + 2^{|K^I|+2|K^O|}$ (Matsui, 1994)

The Walsh Transform Technique

(Collard, Standaert, Quisquater, 2007), (Flórez-Gutiérrez, Naya-Plasencia, 2020)

$$\widehat{\mathrm{cor}}(K^O, K^I) = \frac{1}{N} \sum_{X} (-1)^{f(X \oplus K^O, K^I)} \underbrace{\sum_{x \in \mathcal{D},\ x \mapsto X} (-1)^{f_0(x)}}_{A[X] \text{ (distillation table)}}$$

The Walsh Transform Technique

(Collard, Standaert, Quisquater, 2007), (Flórez-Gutiérrez, Naya-Plasencia, 2020)

$$\widehat{\mathrm{cor}}(K^O, K^I) = \frac{1}{N}\sum_{X}(-1)^{f(X\oplus K^O, K^I)} \underbrace{\sum_{x\in\mathcal{D},\ x\mapsto X}(-1)^{f_0(x)}}_{A[X]\text{ (distillation table)}}$$

Because of the convolution theorem, we can compute $\widehat{\mathrm{cor}}$ as follows:

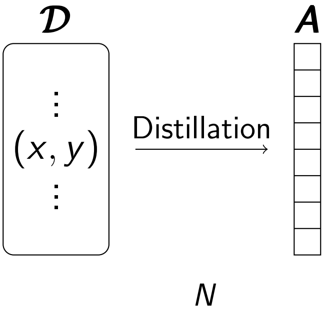
- Compute the distillation table A from the data (N additions)
- Apply the fast Walsh transform on A ($|K^O|2^{|K^O|}$ additions)
- For each guess of K^I ($2^{|K^I|}$ in total):
  - Multiply elementwise by the Walsh spectrum of $f(\,\cdot\,, K^I)$ ($2^{|K^O|}$ products)
  - Apply the fast Walsh transform ($|K^O|2^{|K^O|}$ additions)

The Walsh Transform Technique *(cont.)*

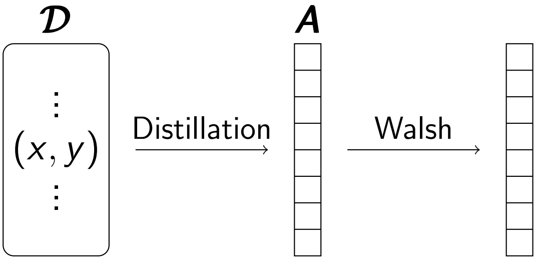
$$\mathcal{D}$$

$$\boxed{\begin{matrix} \vdots \\ (x, y) \\ \vdots \end{matrix}}$$

The Walsh Transform Technique *(cont.)*

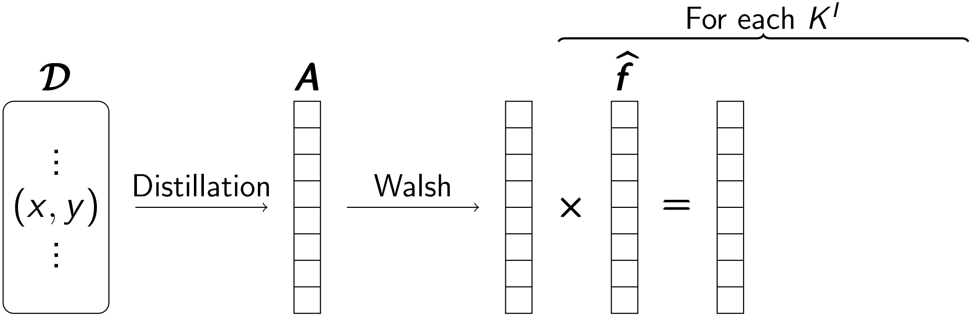


The Walsh Transform Technique *(cont.)*



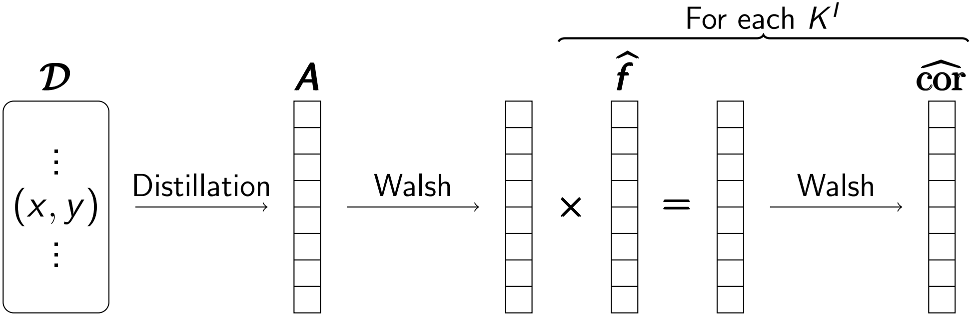
$$N \quad + \quad |K^O| 2^{|K^O|}$$

The Walsh Transform Technique *(cont.)*



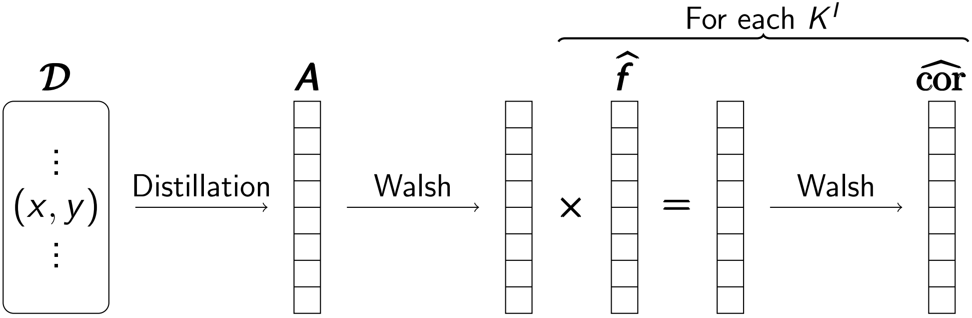
$$N \quad + \quad |K^O|2^{|K^O|} \quad + \quad 2^{|K^I|+|K^O|}$$

The Walsh Transform Technique *(cont.)*



$$N \quad + \quad |K^O| 2^{|K^O|} \quad + \quad 2^{|K^I| + |K^O|} \quad + \quad |K^O| 2^{|K^I| + |K^O|}$$

The Walsh Transform Technique *(cont.)*



$$N \quad + \quad |K^O|2^{|K^O|} \quad + \quad 2^{|K^I|+|K^O|} \quad + \quad |K^O|2^{|K^I|+|K^O|}$$

Total complexity: $\mathcal{O}(N) + \mathcal{O}\left(|K^O|2^{|K^O|+|K^I|}\right)$

The Target Problem

In the cryptanalysis of real ciphers, there can be exploitable redundancies:

- If $N < 2^{|K^O|}$, then A is a sparse vector $\rightarrow$ Do we really need to fully build it?
- Given K^I, the key schedule may make some guesses of K^O impossible $\rightarrow$ Can we avoid computing the associated entries of $\widehat{\mathrm{cor}}$?
- The attack treats the key recovery map as an arbitrary Boolean function $\rightarrow$ Can we exploit any specific properties to improve the time complexity?

The Target Problem

In the cryptanalysis of real ciphers, there can be exploitable redundancies:

- If $N < 2^{|K^O|}$, then A is a sparse vector $\rightarrow$ Do we really need to fully build it?
- Given K^I, the key schedule may make some guesses of K^O impossible $\rightarrow$ Can we avoid computing the associated entries of $\widehat{\mathrm{cor}}$?
- The attack treats the key recovery map as an arbitrary Boolean function $\rightarrow$ Can we exploit any specific properties to improve the time complexity?

These redundancies can be expressed as sparsity properties of the nonzero inputs and desired outputs of the Walsh transform steps

The Target Problem

In the cryptanalysis of real ciphers, there can be exploitable redundancies:

- If $N < 2^{|K^O|}$, then A is a sparse vector $\rightarrow$ Do we really need to fully build it?
- Given K^I, the key schedule may make some guesses of K^O impossible $\rightarrow$ Can we avoid computing the associated entries of $\widehat{\mathrm{cor}}$?
- The attack treats the key recovery map as an arbitrary Boolean function $\rightarrow$ Can we exploit any specific properties to improve the time complexity?

These redundancies can be expressed as sparsity properties of the nonzero inputs and desired outputs of the Walsh transform steps

Problem: The fast Walsh transform algorithm is a box with fixed time complexity

Structure of the Presentation

1. Affine Walsh Transform Pruning
   - Problem statement and complexity result
   - Example of pruned fast Walsh transform algorithm

Structure of the Presentation

1. Affine Walsh Transform Pruning
    - Problem statement and complexity result
    - Example of pruned fast Walsh transform algorithm
2. Assembling the Attack
    - Walsh spectrum sparsity properties
    - Improving the second Walsh transform step
    - Improving the first Walsh transform step

Structure of the Presentation

1. Affine Walsh Transform Pruning
    - Problem statement and complexity result
    - Example of pruned fast Walsh transform algorithm
2. Assembling the Attack
    - Walsh spectrum sparsity properties
    - Improving the second Walsh transform step
    - Improving the first Walsh transform step
3. Applications and Conclusion
    - Application to the DES
    - Application to 29-round PRESENT-128
    - Open problems

Affine Walsh Transform Pruning

Problem Statement and Complexity Result

Affine Pruning Problem Statement

Consider a vector of length 2^n $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ and:

- A list $L \subseteq \mathbb{F}_2^n$ of inputs so that $f(x) = 0$ if $x \notin L$
- An affine subspace $x_0 + X \subseteq \mathbb{F}_2^n$ so that $L \subseteq x_0 + X$
- A list $M \subseteq \mathbb{F}_2^n$ of outputs
- An affine subspace $u_0 + U \subseteq \mathbb{F}_2^n$ so that $M \subseteq u_0 + U$

We wish to compute $\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle x,u \rangle} f(x)$ for all $u \in M$ efficiently

Problem Statement and Complexity Result

Affine Pruning Problem Statement

Consider a vector of length 2^n $f : \mathbb{F}_2^n \longrightarrow \mathbb{C}$ and:

- A list $L \subseteq \mathbb{F}_2^n$ of inputs so that $f(x) = 0$ if $x \notin L$
- An affine subspace $x_0 + X \subseteq \mathbb{F}_2^n$ so that $L \subseteq x_0 + X$
- A list $M \subseteq \mathbb{F}_2^n$ of outputs
- An affine subspace $u_0 + U \subseteq \mathbb{F}_2^n$ so that $M \subseteq u_0 + U$

We wish to compute $\widehat{f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle x,u \rangle} f(x)$ for all $u \in M$ efficiently

Theorem: Walsh Transform Affine Pruning Algorithm

The previous problem can be solved in $|L| + t2^t + |M|$ operations, where

$$t = \dim\left(X/(X \cap U^\perp)\right) = \dim\left(U/(U \cap X^\perp)\right)$$

Affine Pruning Example

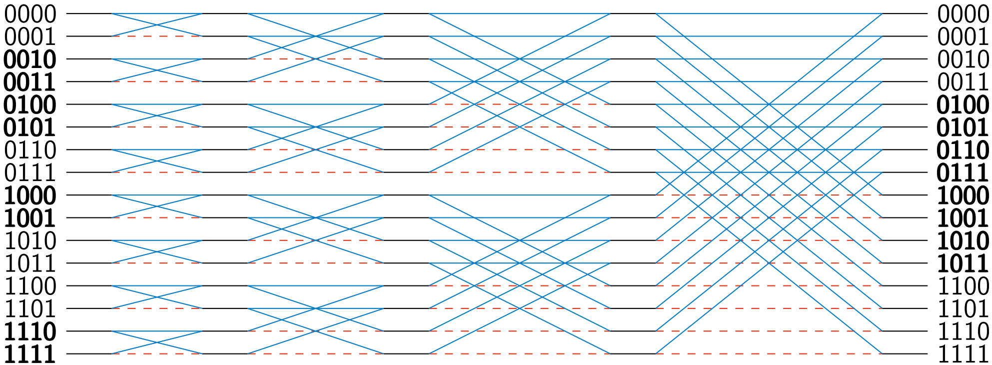
$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

0000	0000
0001	0001
0010	0010
0011	0011
0100	**0100**
0101	**0101**
0110	**0110**
0111	**0111**
1000	**1000**
1001	**1001**
1010	**1010**
1011	**1011**
1100	1100
1101	1101
1110	1110
1111	1111

We try different strategies:

Affine Pruning Example

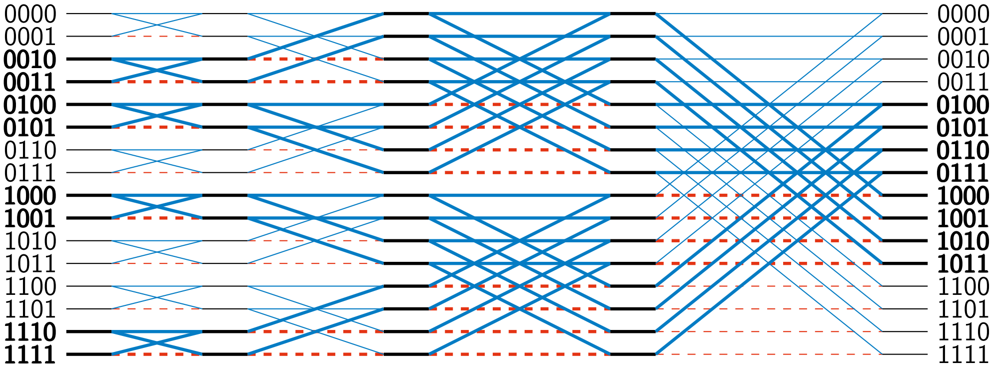
$$\begin{cases} x_0 + X = (0010) + \mathrm{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \mathrm{span}\{(0001),(0010),(1100)\} \end{cases}$$



We try different strategies: 64 operations

Affine Pruning Example

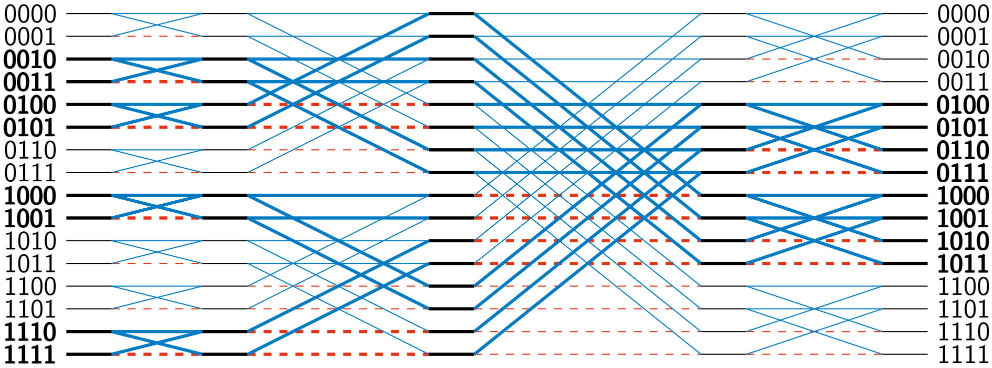
$$\begin{cases} x_0 & + & X & = & (0010) & + & \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$



We try different strategies: 40 operations

Affine Pruning Example

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001), (0110), (1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001), (0010), (1100)\} \end{cases}$$



We try different strategies: 32 operations

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 & + & X & = & (0010) & + & \mathrm{span}\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \mathrm{span}\{(0001),(0010),(1100)\} \end{cases}$$

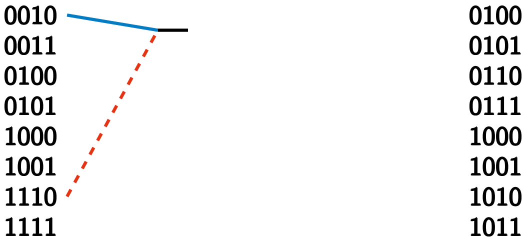
$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

0010	**0100**
0011	**0101**
0100	**0110**
0101	**0111**
1000	**1000**
1001	**1001**
1110	**1010**
1111	**1011**

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

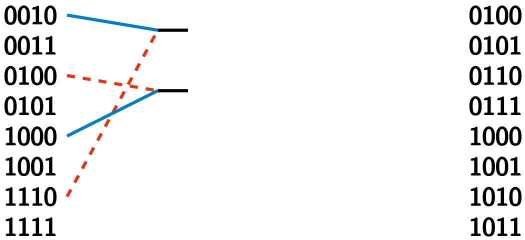


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001), (0110), (1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001), (0010), (1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

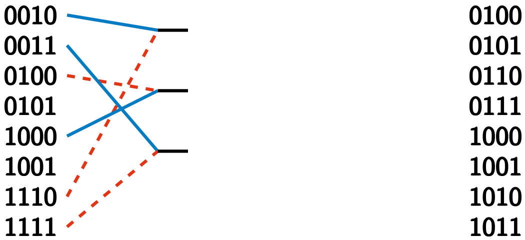


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 & + & X & = & (0010) & + & \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

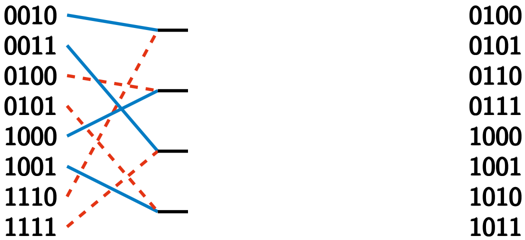


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 & + & X & = & (0010) & + & \mathrm{span}\,\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \mathrm{span}\,\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

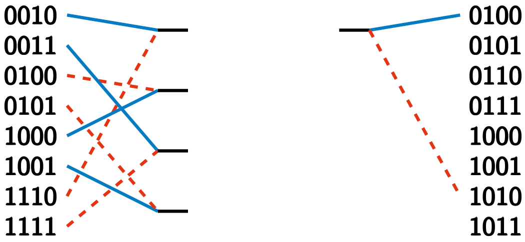


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 & + & X & = & (0010) & + & \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} \color{blue}{1} & \color{blue}{1} & \color{blue}{-1} & \color{blue}{-1} & \color{blue}{1} & \color{blue}{1} & \color{blue}{-1} & \color{blue}{-1} \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ \color{red}{-1} & \color{red}{-1} & \color{red}{1} & \color{red}{1} & \color{red}{-1} & \color{red}{-1} & \color{red}{1} & \color{red}{1} \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

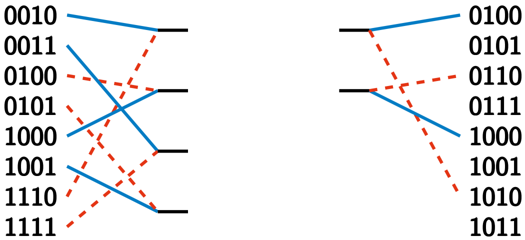


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs
- Outputs differing by $(1110) \in X^{\perp}$ are opposites

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 & + & X & = & (0010) & + & \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 & + & U & = & (0100) & + & \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

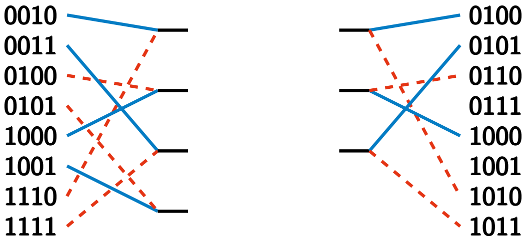


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs
- Outputs differing by $(1110) \in X^{\perp}$ are opposites

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

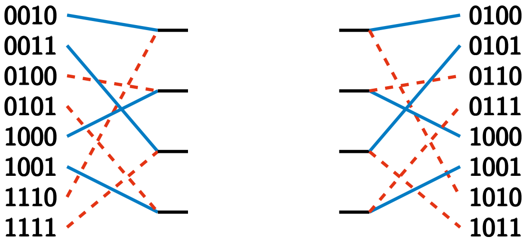


- Inputs differing by $(1100) \in U^\perp$ appear with opposite signs
- Outputs differing by $(1110) \in X^\perp$ are opposites

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

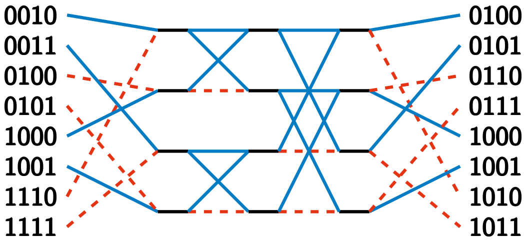


- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs
- Outputs differing by $(1110) \in X^{\perp}$ are opposites

Affine Pruning Example *(cont.)*

$$\begin{cases} x_0 + X = (0010) + \operatorname{span}\{(0001),(0110),(1010)\} \\ u_0 + U = (0100) + \operatorname{span}\{(0001),(0010),(1100)\} \end{cases}$$

$$H = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix}$$



- Inputs differing by $(1100) \in U^{\perp}$ appear with opposite signs
- Outputs differing by $(1110) \in X^{\perp}$ are opposites
- The transform is reduced to one of size 4

16 operations

Overview of the General Algorithm

The complexity is determined by the dimensions of X and U and their orthogonality: The more "orthogonal" X and U are, the lower the complexity, as

$$t = \dim(X) - \dim\left(X \cap U^{\perp}\right) = \dim(U) - \dim\left(U \cap X^{\perp}\right)$$

Overview of the General Algorithm

The complexity is determined by the dimensions of X and U and their orthogonality: The more "orthogonal" X and U are, the lower the complexity, as

$$t = \dim(X) - \dim\left(X \cap U^{\perp}\right) = \dim(U) - \dim\left(U \cap X^{\perp}\right)$$

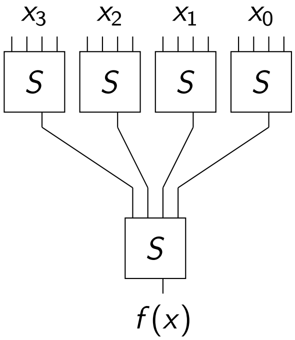
The pruned algorithm consists of three steps:

1. A compression step which adds/subtracts each input to a position in a vector of length 2^t, with cost $\mathcal{O}(1)$ for each nonzero input
2. A fast Walsh transform on this compressed vector, with cost $t2^t$
3. An expansion step which maps each desired output to a position in this vector, with cost $\mathcal{O}(1)$ for each desired output

Assembling the Attack

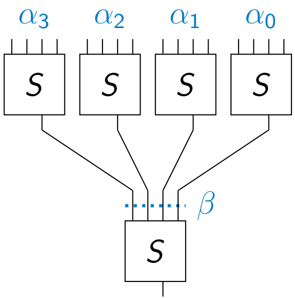
Walsh Spectrum Example

We assume S balanced and consider the following map:



Walsh Spectrum Example

We assume S balanced and consider the following map:



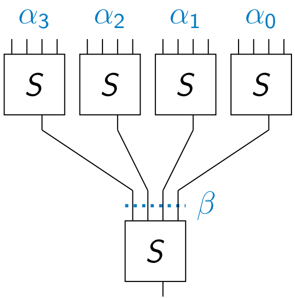
We have a nice formula for its Walsh coefficients:

$$\hat{f}(\alpha_3, \alpha_2, \alpha_1, \alpha_0) = \frac{1}{4} \hat{S}(\alpha_3)\,\hat{S}(\alpha_2)\,\hat{S}(\alpha_1)\,\hat{S}(\alpha_0)\,\hat{S}(\beta),$$

where $\beta_i = 1 \Leftrightarrow \alpha_i \neq 0$ because S is balanced

Walsh Spectrum Example

We assume S balanced and consider the following map:



We have a nice formula for its Walsh coefficients:

$$\hat{f}(\alpha_3, \alpha_2, \alpha_1, \alpha_0) = \frac{1}{4} \hat{S}(\alpha_3) \hat{S}(\alpha_2) \hat{S}(\alpha_1) \hat{S}(\alpha_0) \hat{S}(\beta),$$

where $\beta_i = 1 \Leftrightarrow \alpha_i \neq 0$ because S is balanced

If $\hat{S}(\mathtt{F}) = 0$, then $\hat{f}(\alpha_3, \alpha_2, \alpha_1, \alpha_0) \neq 0 \implies \alpha_i = 0$ for some i

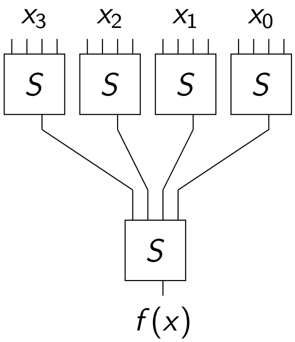
The nonzero Walsh coefficients of f are contained in 4 vector subspaces of dimension 12 of $\mathbb{F}_2^{16}$, given by the conditions $\alpha_0 = 0$, $\alpha_1 = 0$, $\alpha_2 = 0$ and $\alpha_3 = 0$

Walsh Spectrum Example *(cont.)*

There are still two issues that we wish to solve:

- What if $\hat{S}(\mathtt{F}) \neq 0$?

  We choose a subset $\mathcal{X}$ so that $\hat{S}_{y\in\mathcal{X}}(\mathtt{F}) = 0$ and reject some of the data accordingly



Walsh Spectrum Example *(cont.)*

There are still two issues that we wish to solve:

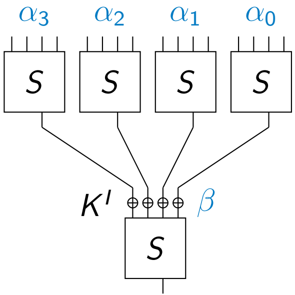
- What if $\hat{S}(\mathtt{F}) \neq 0$?

  We choose a subset $\mathcal{X}$ so that $\hat{S}_{y \in \mathcal{X}}(\mathtt{F}) = 0$ and reject some of the data accordingly

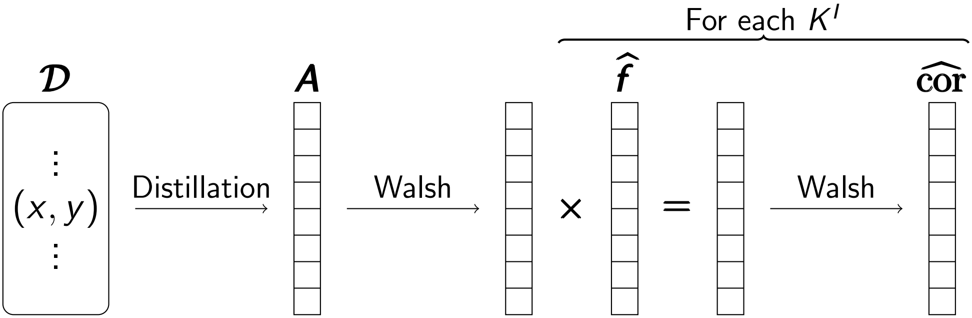
- What happens with the addition of K^I?

$$\hat{f}_{K^I}(\alpha_3, \alpha_2, \alpha_1, \alpha_0) \;=\; (-1)^{\langle K^I, \beta \rangle} \hat{f}_0(\alpha_3, \alpha_2, \alpha_1, \alpha_0)$$

  - The vector subspaces are the same for all K^I
  - We can compute $\hat{f}_0$ and deduce the spectrum of other K^I with sign swaps

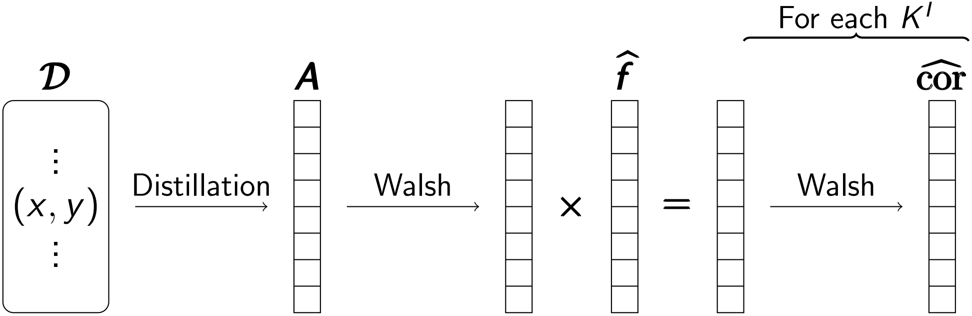


Improving the Second Walsh Transform Step



$$N \quad + \quad |K^O|2^{|K^O|} \quad + \quad 2^{|K^I|+|K^O|} \quad + \quad |K^O|2^{|K^I|+|K^O|}$$

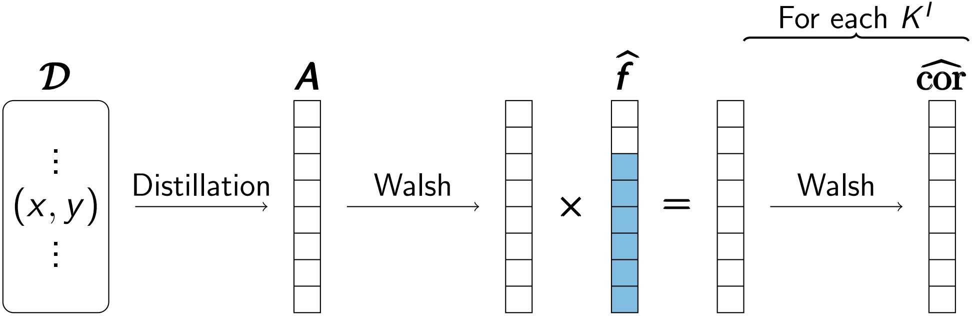
Improving the Second Walsh Transform Step



$$N \quad + \quad |K^O| 2^{|K^O|} \quad + \quad 2^{|K^O|} \quad + \quad |K^O| 2^{|K^I| + |K^O|}$$

- We can multiply by the Walsh spectrum associated to $K^I = 0$ and sign swap at the start of the second Walsh transform

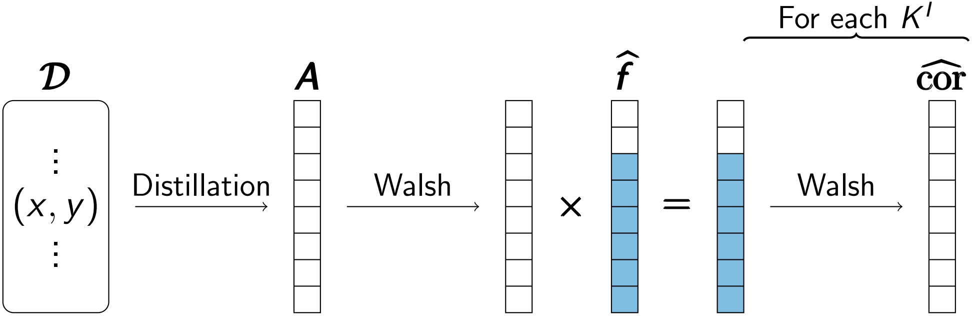
Improving the Second Walsh Transform Step



$$N \quad + \quad |K^O|2^{|K^O|} \quad + \quad 2^{|K^O|} \quad + \quad |K^O|2^{|K^I|+|K^O|}$$

- We can multiply by the Walsh spectrum associated to $K^I = 0$ and sign swap at the start of the second Walsh transform
- We next look at the support of the Walsh spectrum of f

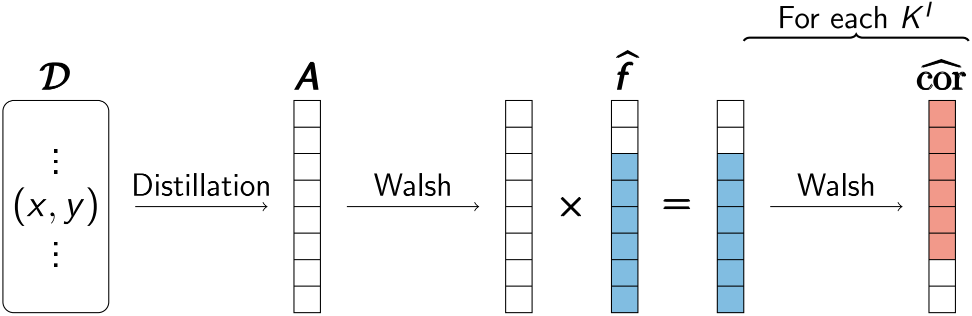
Improving the Second Walsh Transform Step *(cont.)*



$$N \quad + \quad |K^O| 2^{|K^O|} \quad + \quad 2^{\dim(U)} \quad + \quad |K^O| 2^{|K^I| + |K^O|}$$

- The nonzero inputs of the second Walsh transform step must be in the support of $\widehat{f}$, which we assume is contained in a subspace U

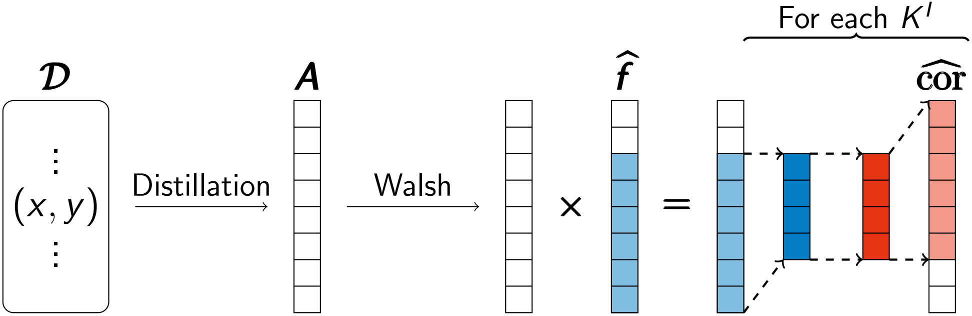
Improving the Second Walsh Transform Step *(cont.)*



$$N + |K^O|2^{|K^O|} + 2^{\dim(U)} + |K^O|2^{|K^I|+|K^O|}$$

- The nonzero inputs of the second Walsh transform step must be in the support of $\widehat{f}$, which we assume is contained in a subspace U
- Given K^I, we assume the possible values of K^O lie in a subspace V

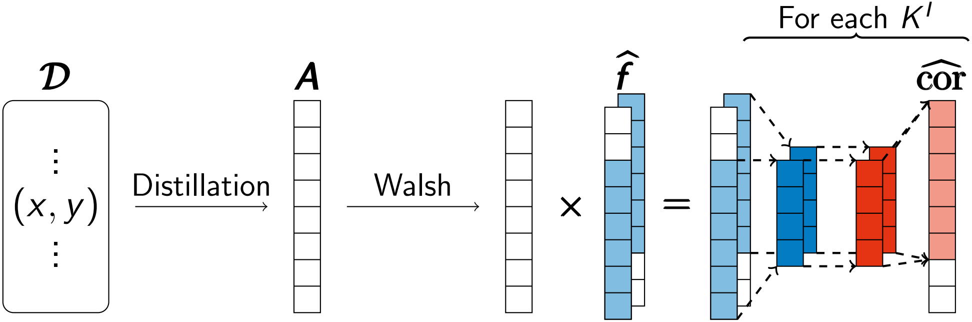
Improving the Second Walsh Transform Step *(cont.)*



$$N \quad + \quad |K^O| 2^{|K^O|} + \quad 2^{\dim(U)} + 2^{|K^I|} \left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)$$

- The transforms are reduced to size 2^r, where $r = \dim(U) - \dim(U \cap V^{\perp})$

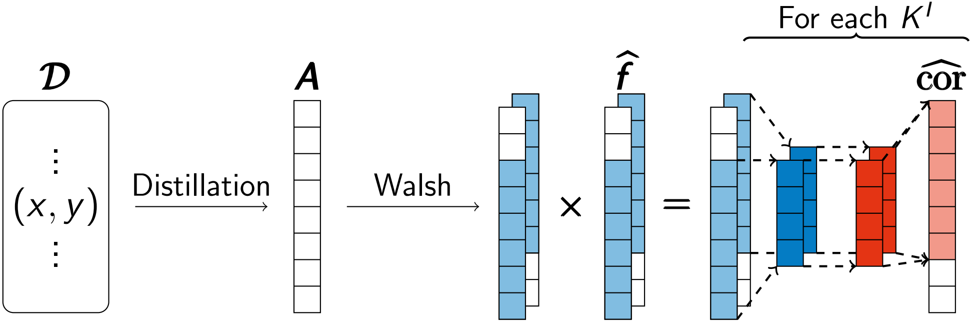
Improving the Second Walsh Transform Step *(cont.)*



$$N \quad + \quad |K^O|2^{|K^O|} + l\left(2^{\dim(U)} + 2^{|K^I|}\left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)\right)$$

- The transforms are reduced to size 2^r, where $r = \dim(U) - \dim(U \cap V^{\perp})$
- If the support of $\widehat{f}$ is covered by l subspaces, we can use the linearity of the Walsh transform to separate it into several parts

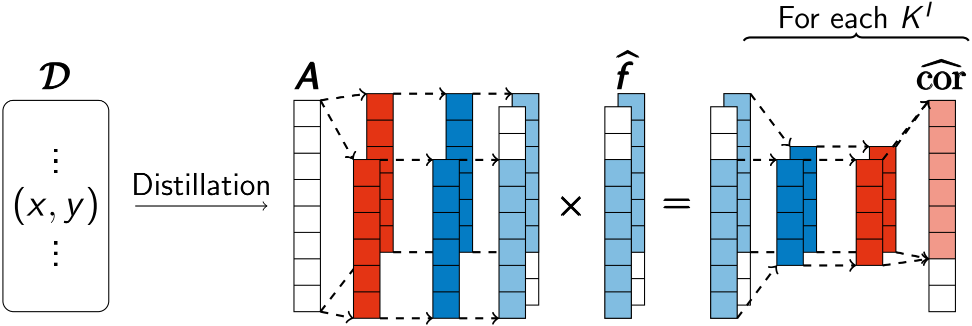
Improving the First Walsh Transform Step



$$N \quad + \quad |K^O| 2^{|K^O|} + I\left(2^{\dim(U)} + 2^{|K^I|}\left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)\right)$$

- We don't need to compute any outputs of the first Walsh transform associated to zeroes in the Walsh spectrum of f

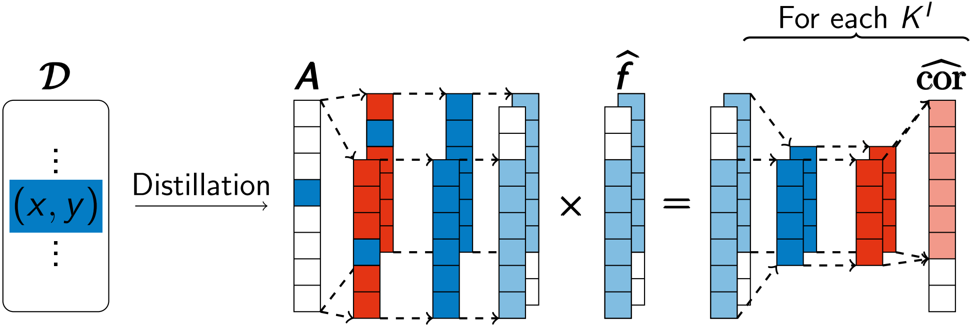
Improving the First Walsh Transform Step



$$N + l\left(2^{|K^O|} + \dim(U)\, 2^{\dim(U)} + 2^{\dim(U)} + 2^{|K^I|}\left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)\right)$$

- We don't need to compute any outputs of the first Walsh transform associated to zeroes in the Walsh spectrum of f
- Which means we can prune the first Walsh transform at the output side

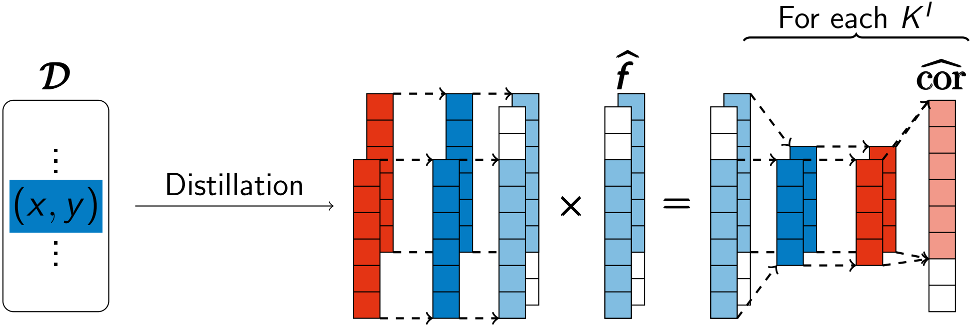
Improving the First Walsh Transform Step *(cont.)*



$$N + I\left(2^{|K^O|} + \dim(U)\, 2^{\dim(U)} + 2^{\dim(U)} + 2^{|K^I|}\left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)\right)$$

- We note that each data pair contributes to exactly one position in A, which then contributes to exactly one position in each of the compressed arrays

Improving the First Walsh Transform Step *(cont.)*



$$I\left(N + \dim(U)\, 2^{\dim(U)} + 2^{\dim(U)} + 2^{|K^I|}\left(2^{\dim(U)} + r2^r + 2^{\dim(V)}\right)\right)$$

- We note that each data pair contributes to exactly one position in A, which then contributes to exactly one position in each of the compressed arrays
- So we can perform the distillation and compression step at the same time, skipping the construction of the array A

Applications and Conclusion

Application to the DES

We propose an attack on the full 16-round DES which is based on (Matsui, 1994)

We cover the last round of Matsui's 14-round approximation with key recovery (one key recovery round at the input and two at the output), and leverage the Walsh spectrum of $S5$ to keep the time complexity down

We achieve the best known attack in terms of data complexity

Type	**Data**	**Time**	**Memory**	**Source**
Differential	$2^{47.00}$ CP	$2^{37.00}$	$\mathcal{O}(1)$	(Biham and Shamir, 1992)
Linear	$2^{43.00}$ KP	$2^{39.00}$	$2^{26.00}$	(Matsui, 1994)
Multiple Linear	$2^{42.78}$ KP	$2^{38.86}$	$2^{30.00}$	(Bogdanov and Vejre, 2017)
Conditional Linear	$2^{42.00}$ KP	$2^{42.00}$	$2^{28.00}$	(Biham and Perle, 2018)
Linear	$2^{41.50}$ KP	$2^{42.13}$	$2^{38.75}$	(Flórez-Gutiérrez, 2022)

Application to 29-round PRESENT-128

We extend the 28-round multiple linear attack on PRESENT-80 of (Flórez-Gutiérrez and Naya-Plasencia, 2020) by adding an additional key recovery round (two key recovery rounds at the input and three at the output)

Without Walsh transform pruning, the attack costs at least 2^{130} operations per linear approximation, with pruning techniques we manage to keep the cost of the full attack below 2^{128} encryptions

Key	**Rds.**	**Data**	**Time**	**Memory**	**Source**
80	27/31	$2^{63.8}$	$2^{77.3}$	$2^{48.0}$	(Bogdanov et al., 2018)
		$2^{63.4}$	$2^{72.0}$	$2^{44.0}$	(Flórez-Gutiérrez and Naya-Plasencia, 2020)
	28/31	$2^{64.0}$	$2^{77.4}$	$2^{51.0}$	(Flórez-Gutiérrez and Naya-Plasencia, 2020)
128	28/31	$2^{64.0}$	2^{122}	$2^{84.6}$	(Flórez-Gutiérrez and Naya-Plasencia, 2020)
	29/31	$2^{64.0}$	$2^{124.06}$	$2^{99.2}$	(Flórez-Gutiérrez, 2022)

Open Problems

- Further applications: Differential-linear attacks seem to be good candidates
- Development of automatic tools to compute the cost of optimised attacks
- Improved use of memory by taking advantage of sparsity and repetition
- Applicability to more general linear layers