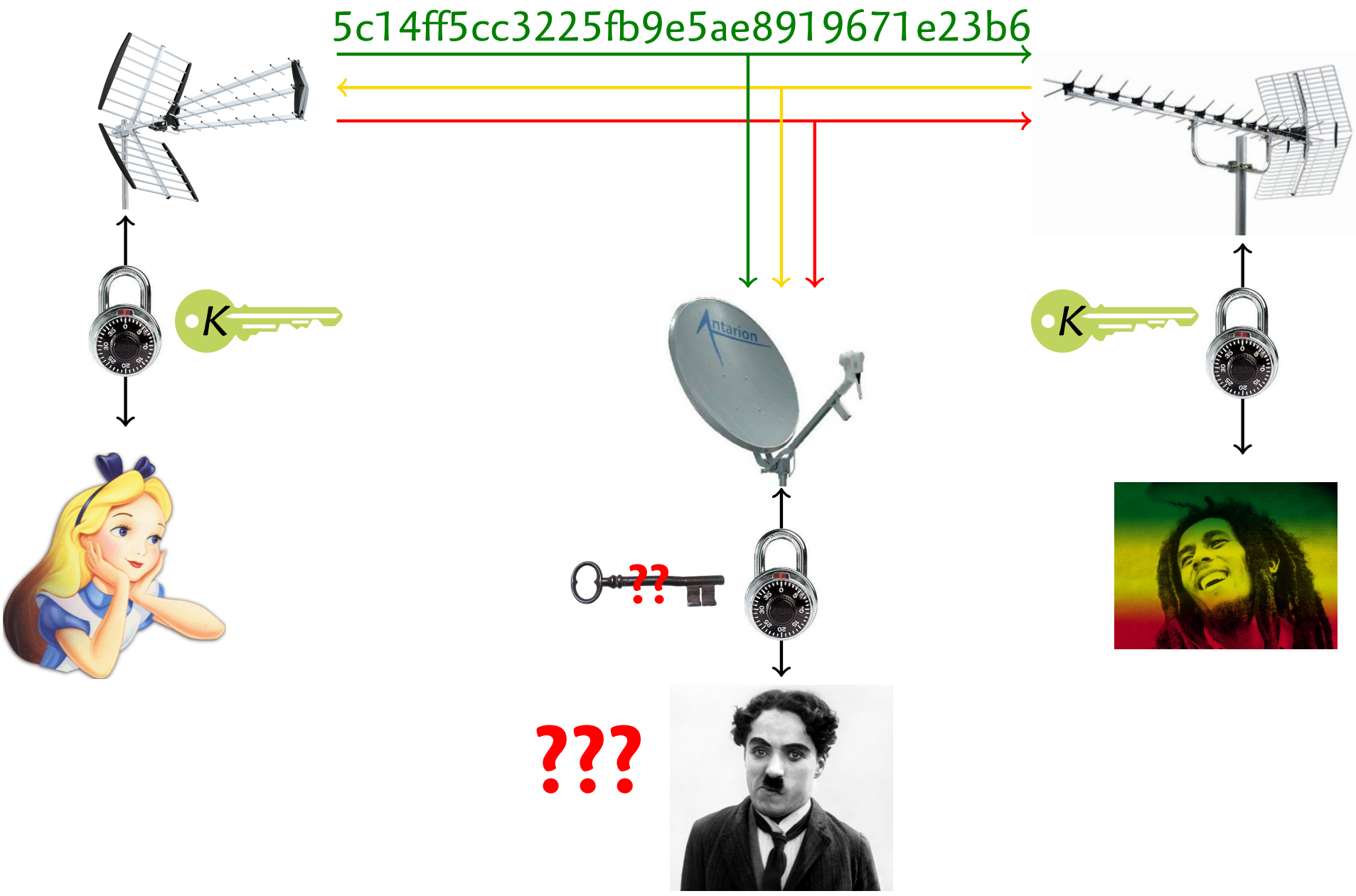
Symbolic Methods for the Automatic Search of Attacks Against Some Block Ciphers

Charles Bouillaguet
(joint work with Patrick Derbez and Pierre-Alain Fouque)

Université de Versailles St-Quentin en Yvelines

LSV Seminar
November 15, 2011

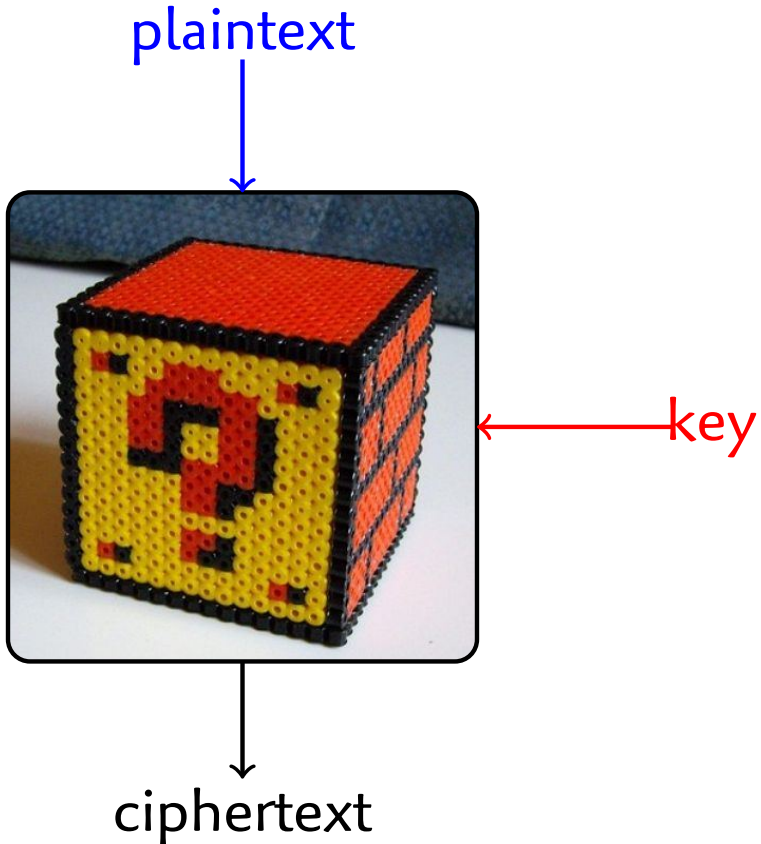
A (Very Brief) Introduction to Cryptography: Encryption



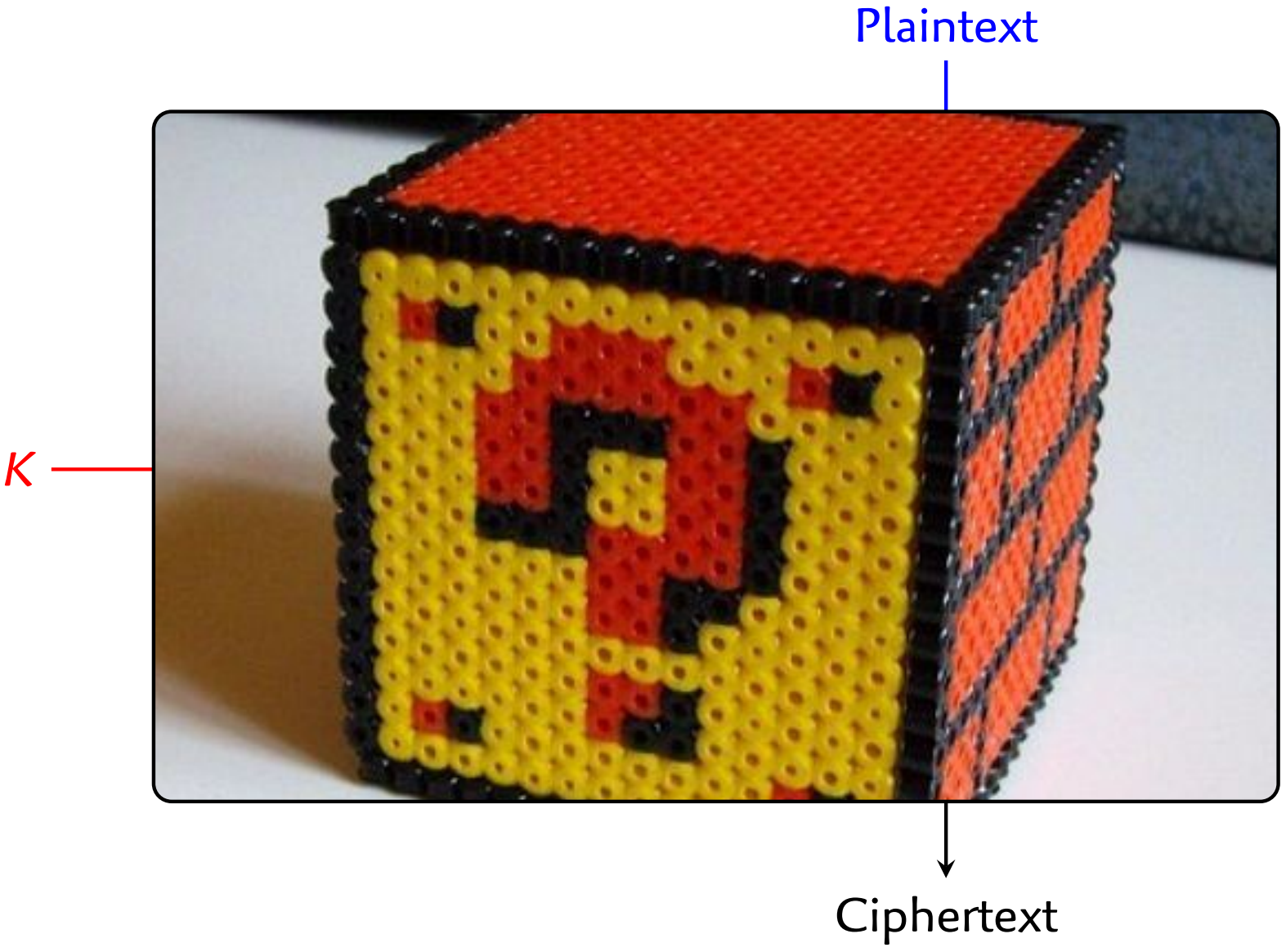
Block-Cipher Cryptanalysis: the Object

a Block Cipher

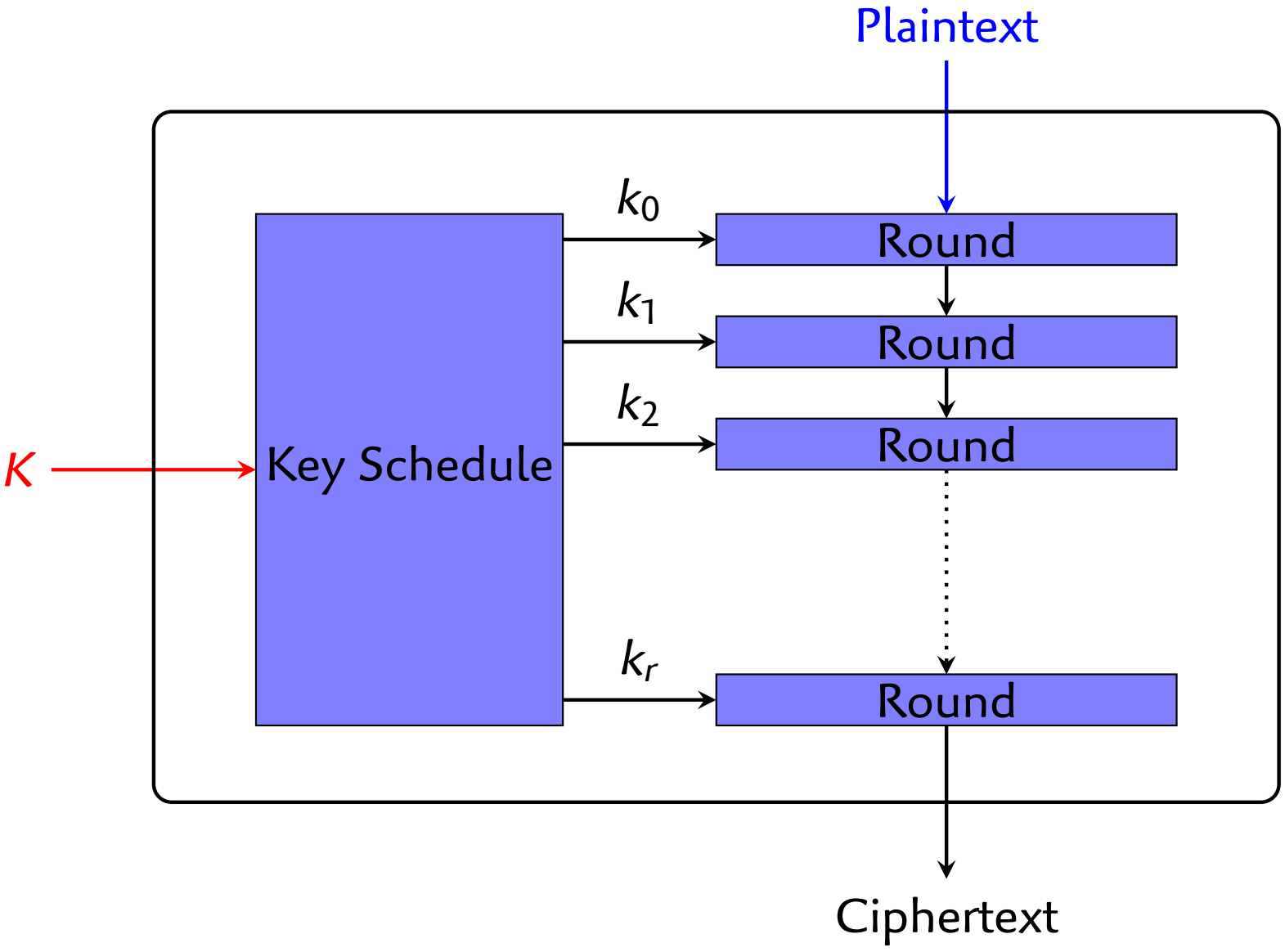
$$E : \underbrace{\{0,1\}^k}_{\text{key}} \times \underbrace{\{0,1\}^n}_{\textit{plaintext}} \to \underbrace{\{0,1\}^n}_{\textit{ciphertext}}$$



Block-Cipher Cryptanalysis: the Object



Block-Cipher Cryptanalysis: the Object

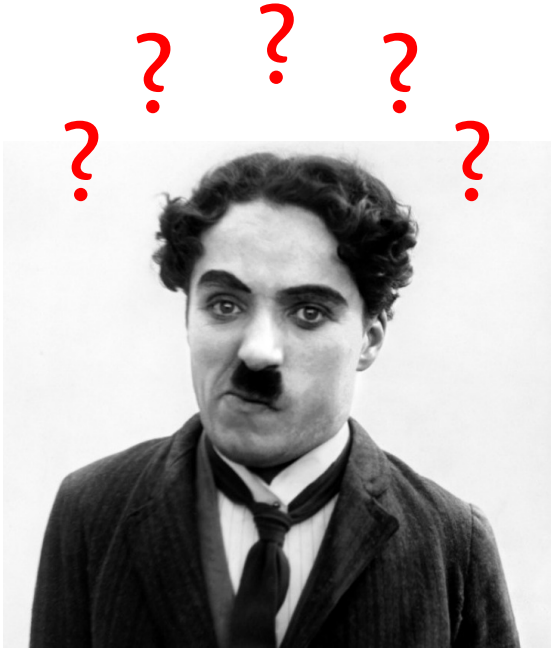


Block-Cipher Cryptanalysis: the Subject

an Attacker

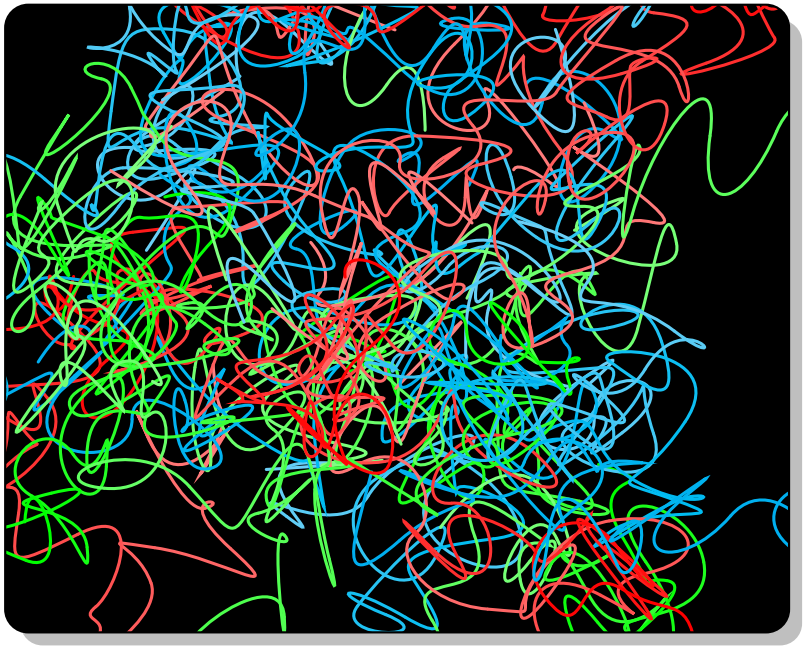
- Goal :
  - In Theory: **distinguish from random permutation**
  - In Practice: **recover the secret key**
- Resources:
  - Time: less than 2^k encryptions
  - Data: less than 2^n plaintext/ciphertext pairs

Block-Cipher Cryptanalysis: the Game

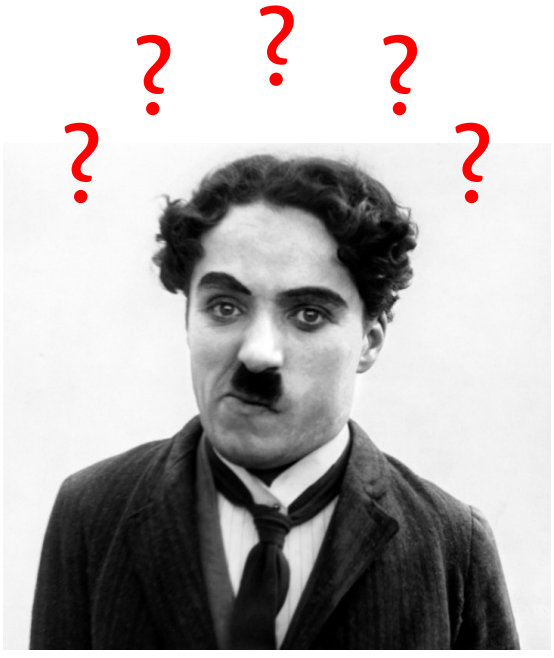


plaintext →

← ciphertext



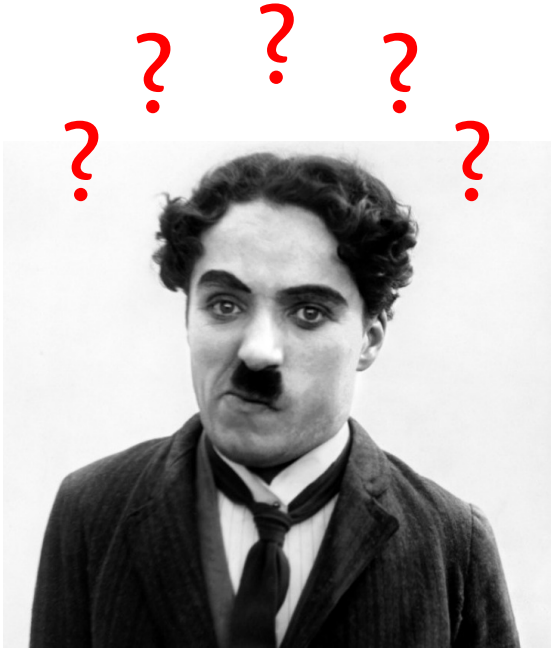
Block-Cipher Cryptanalysis: the Game



plaintext →

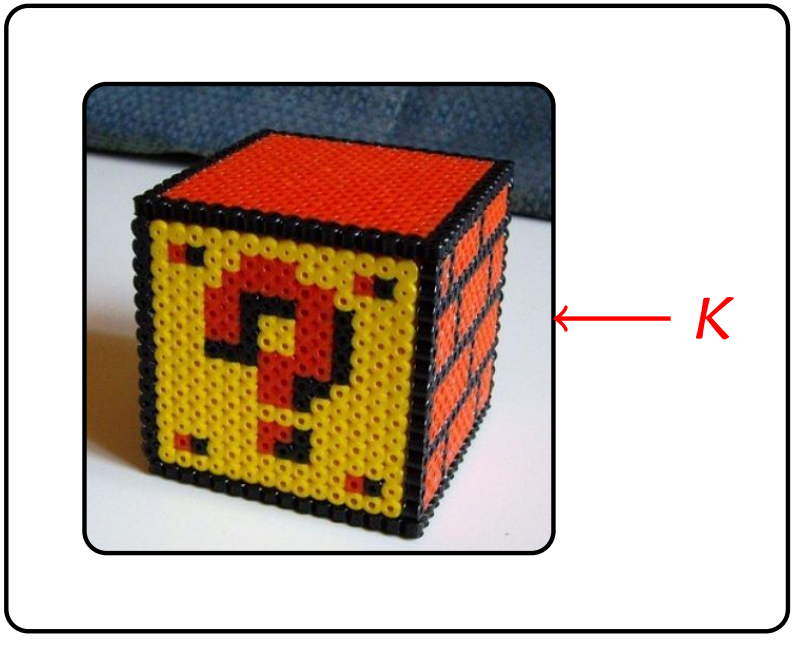
← ciphertext

Block-Cipher Cryptanalysis: the Game



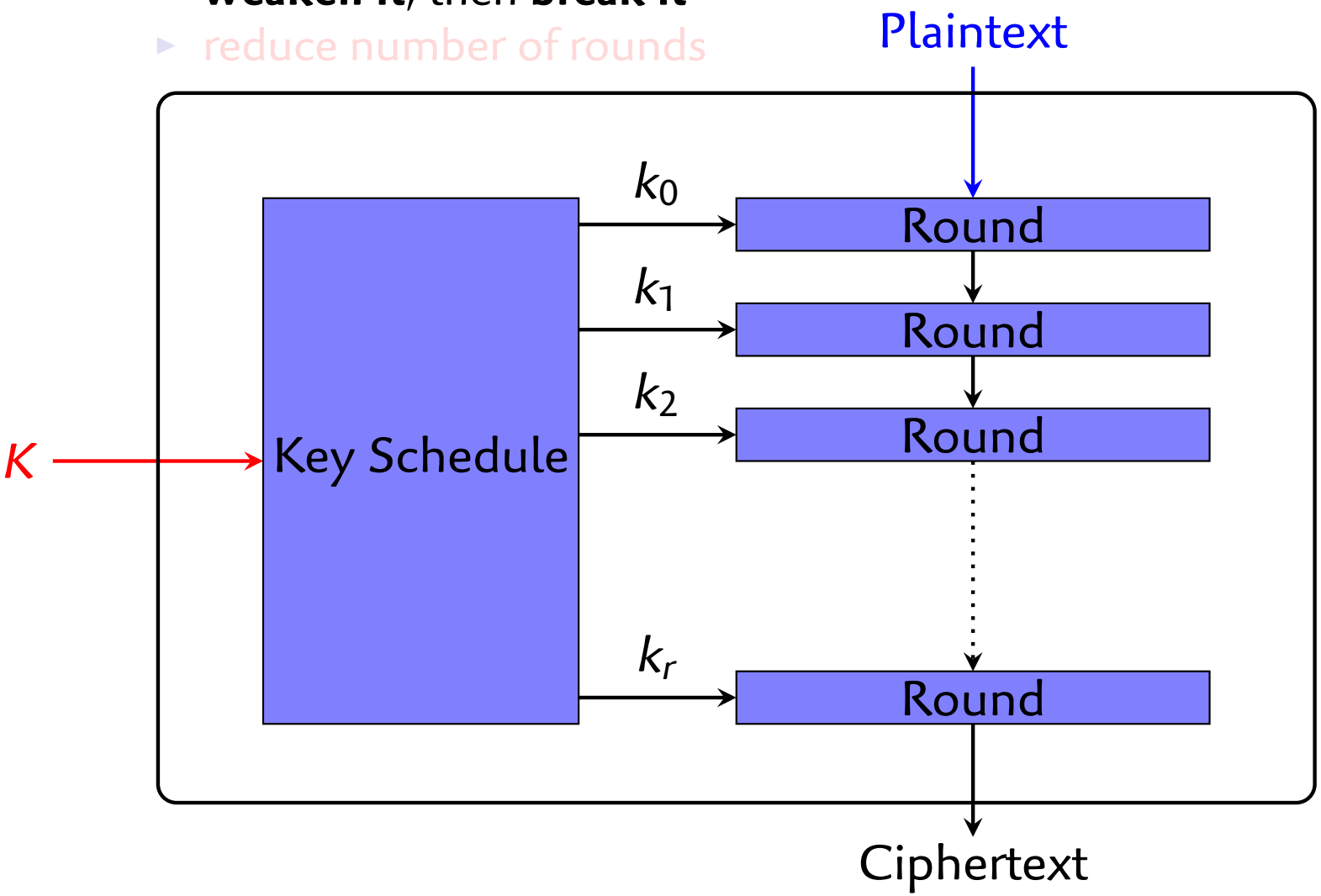
plaintext →

← ciphertext



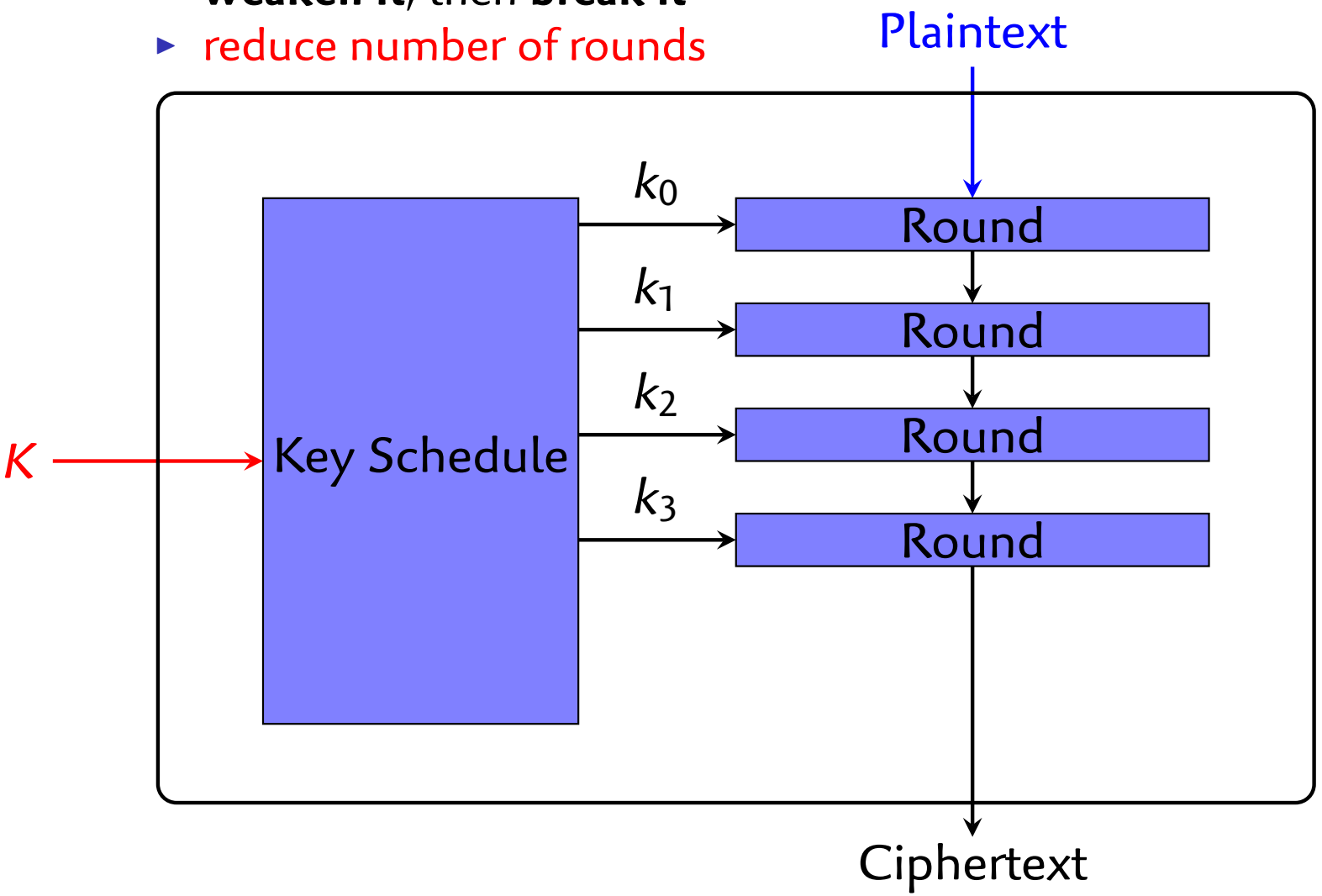
What Can We Do When Block Ciphers Are Too Strong For Us?

- **Solution # 1:**
  - **weaken it**, then **break it**
  - reduce number of rounds



What Can We Do When Block Ciphers Are Too Strong For Us?

- **Solution # 1:**
  - **weaken it**, then **break it**
  - reduce number of rounds

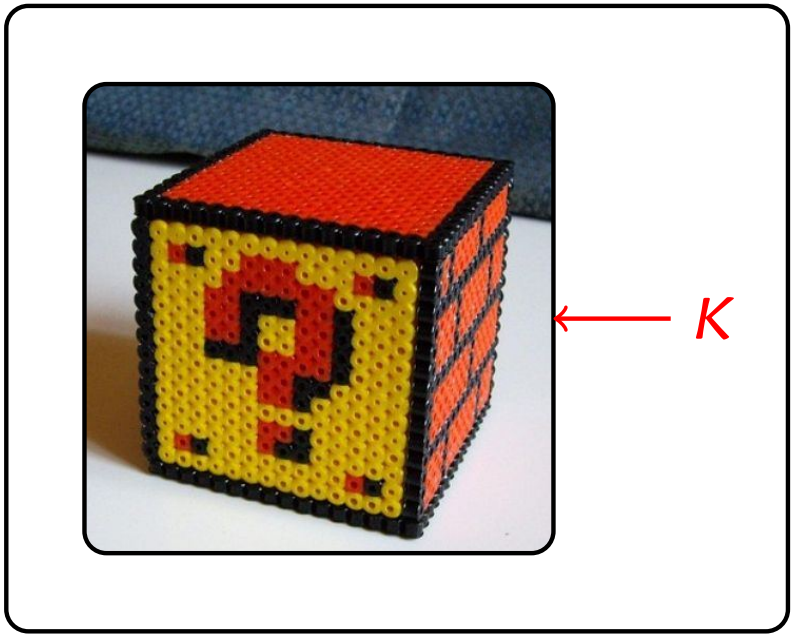


What to do when block ciphers are too strong for us?

- **Solution # 2**:
  - **we get stronger**, then **break it**
  - (adaptively) chosen ciphertexts, related keys, etc.

plaintext →

← ciphertext

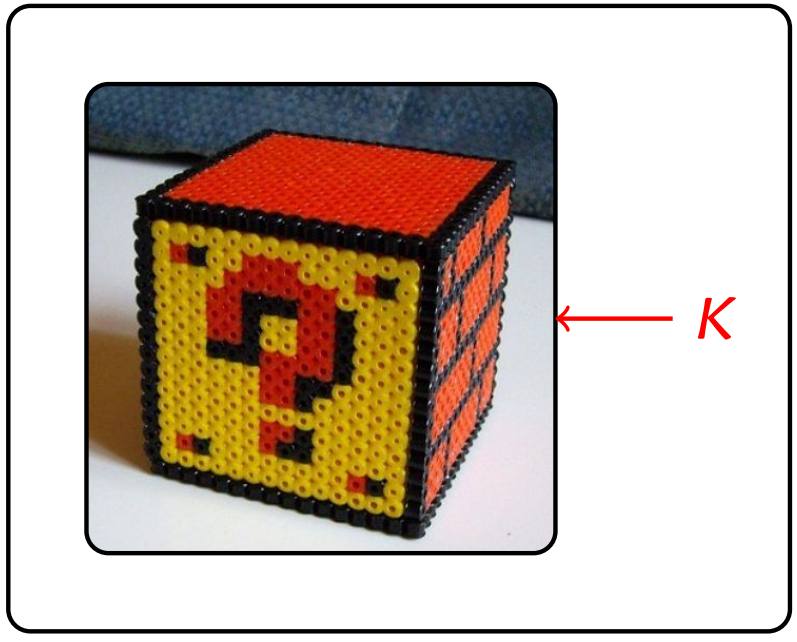


What to do when block ciphers are too strong for us?

- **Solution # 2**:
  - **we get stronger**, then **break it**
  - (adaptively) chosen ciphertexts, related keys, etc.

ciphertext →

← plaintext

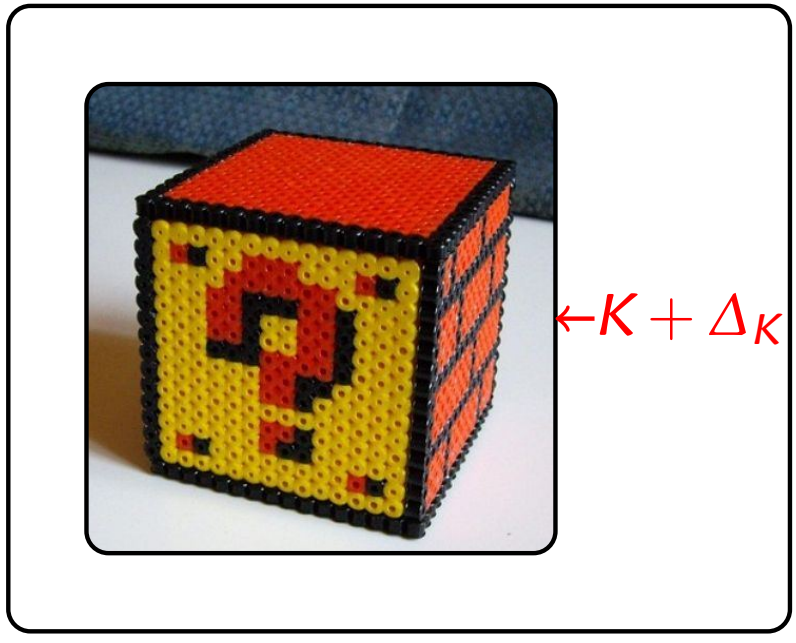


What to do when block ciphers are too strong for us?

- **Solution # 2**:
  - **we get stronger**, then **break it**
  - (adaptively) chosen ciphertexts, related keys, etc.

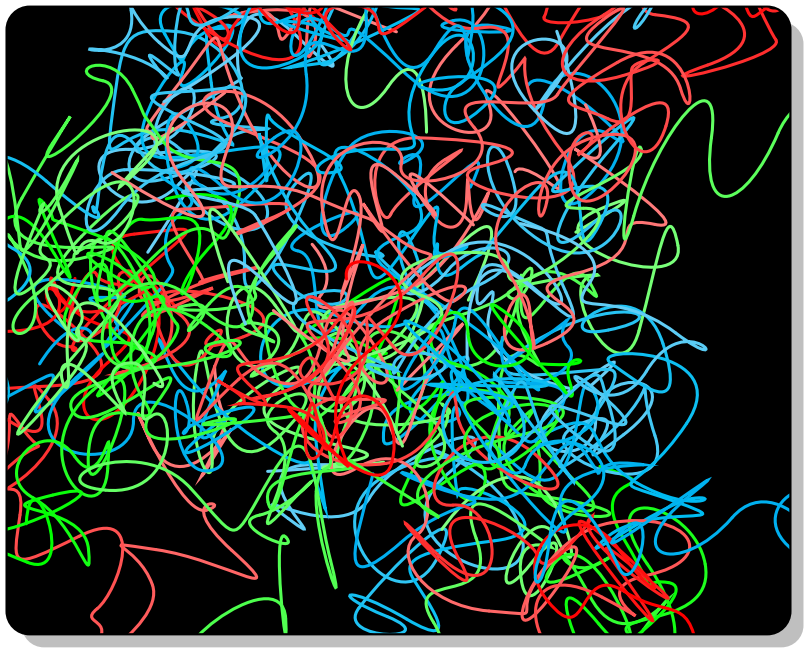
ciphertext →

← plaintext



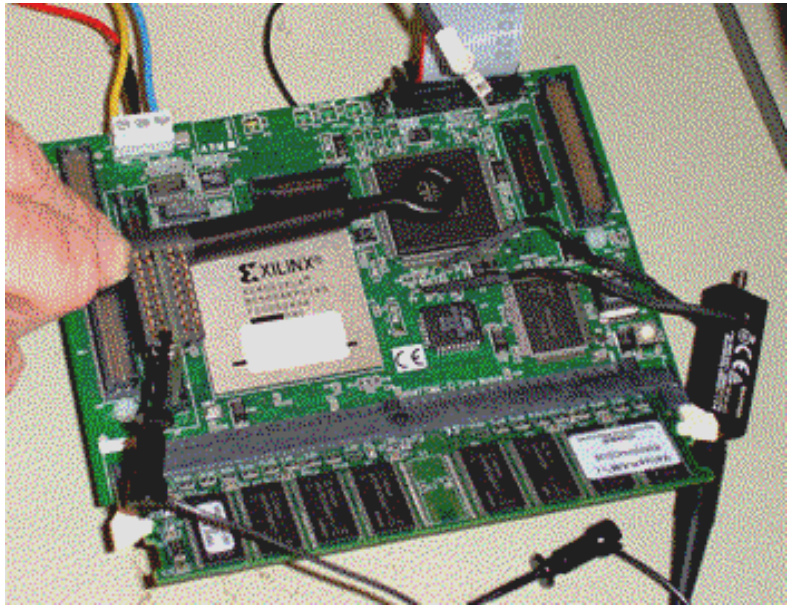
What to do when block ciphers are too strong for us?

- **Solution # 3**:
  - **change the rules**, then **break it**
  - side channels, fault injection, different attack games...



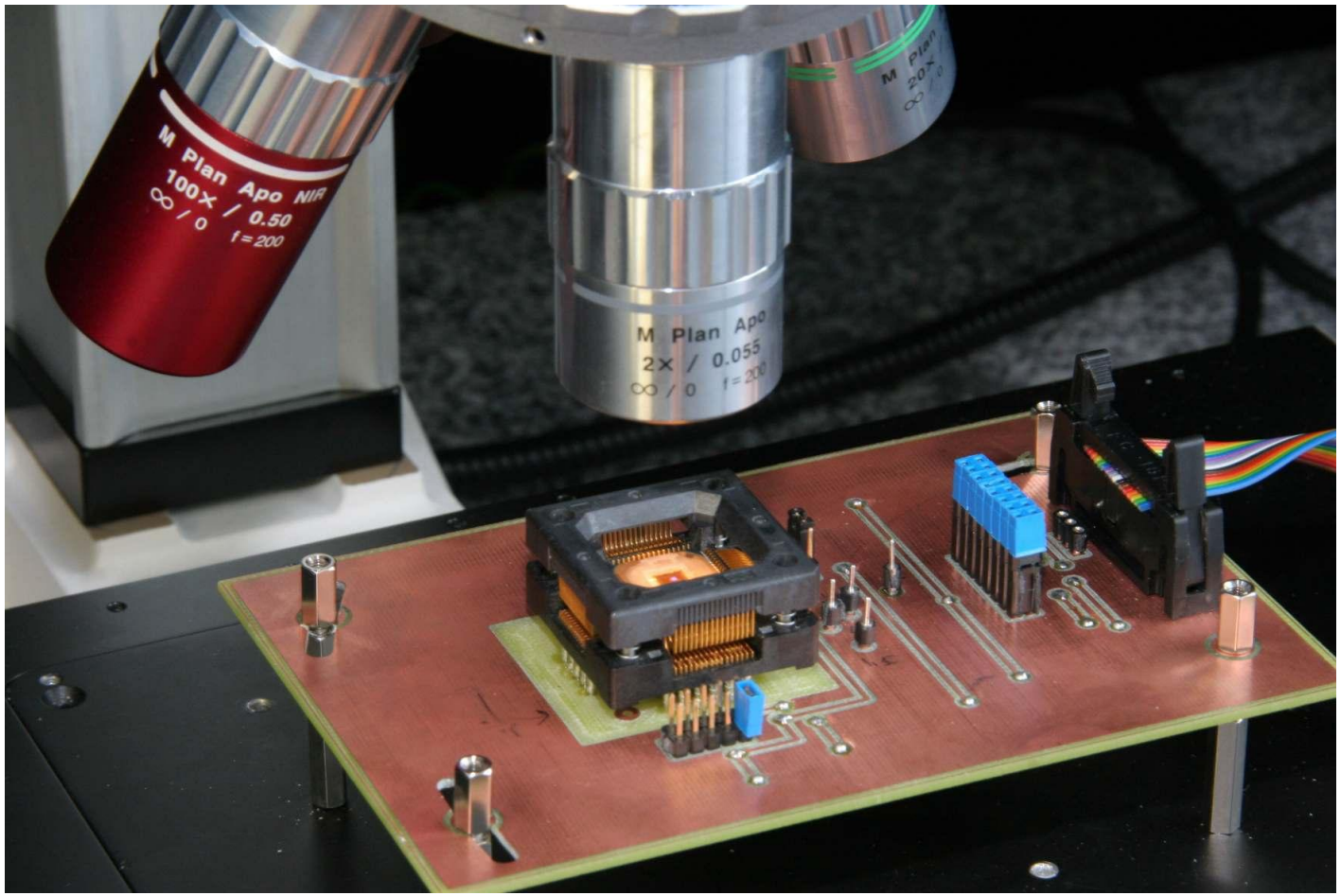
What to do when block ciphers are too strong for us?

- **Solution # 3**:
  - **change the rules**, then **break it**
  - side channels, fault injection, different attack games...



What to do when block ciphers are too strong for us?

- **Solution # 3**:
  - **change the rules**, then **break it**
  - side channels, fault injection, different attack games...



What to do when block ciphers are too strong for us?

- **Solution # 3**:
  - **change the rules**, then **break it**
  - side channels, fault injection, different attack games...

In this talk :

Low Data Complexity Attacks

- Must be faster than exhaustive search
- Only very few plaintext/ciphertext pairs available

Why ?

- Rather unexplored territory
- What is harder in practice?
  - **performing** 2^{50} elementary operations?
  - or **acquiring** 50 Plaintext/Ciphertext pairs?
- LDC attacks can sometimes be recycled, and used as sub-components in other attacks
  - *e.g.* attack on GOST uses a 2-plaintext attack on 8 rounds

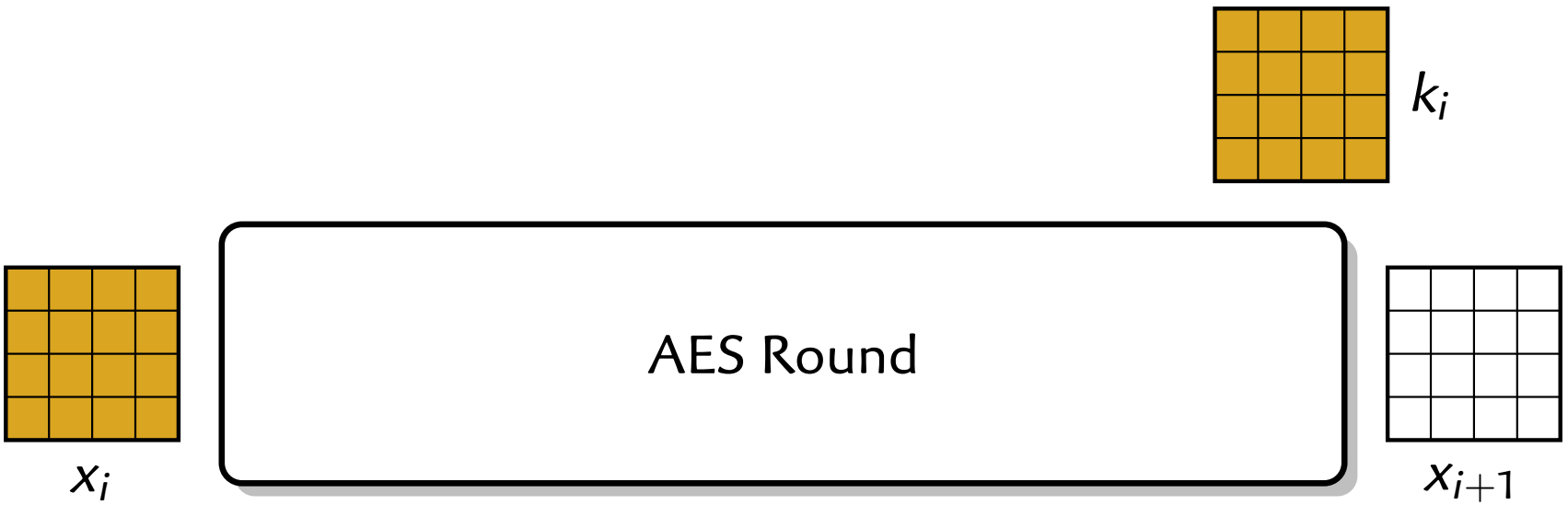
Target Block Cipher: the Advanced Encryption Standard

- Designed by Rijmen and Daemen
- Winner of AES competition in 2001
- One of the most widely used encryption primitive

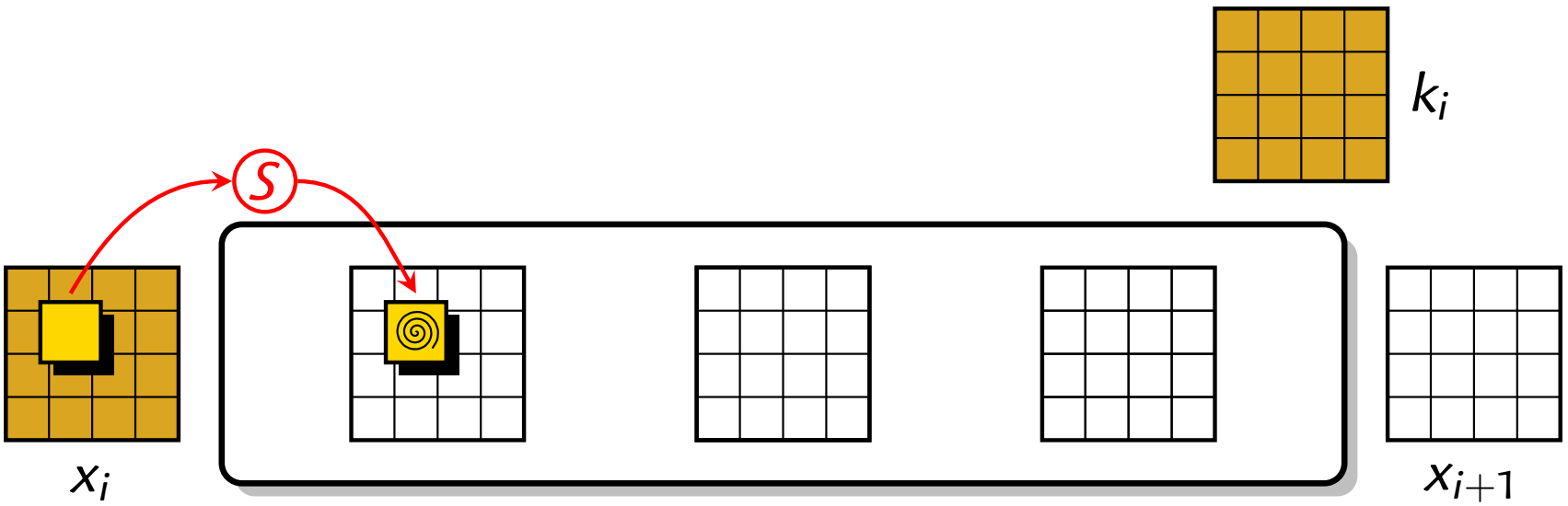
AES basic structures

- Substitution-Permutation network
- Block size: 128 bits
- key lengths: **128**, 192 or 256 bits
- 10 rounds for the 128-bit version

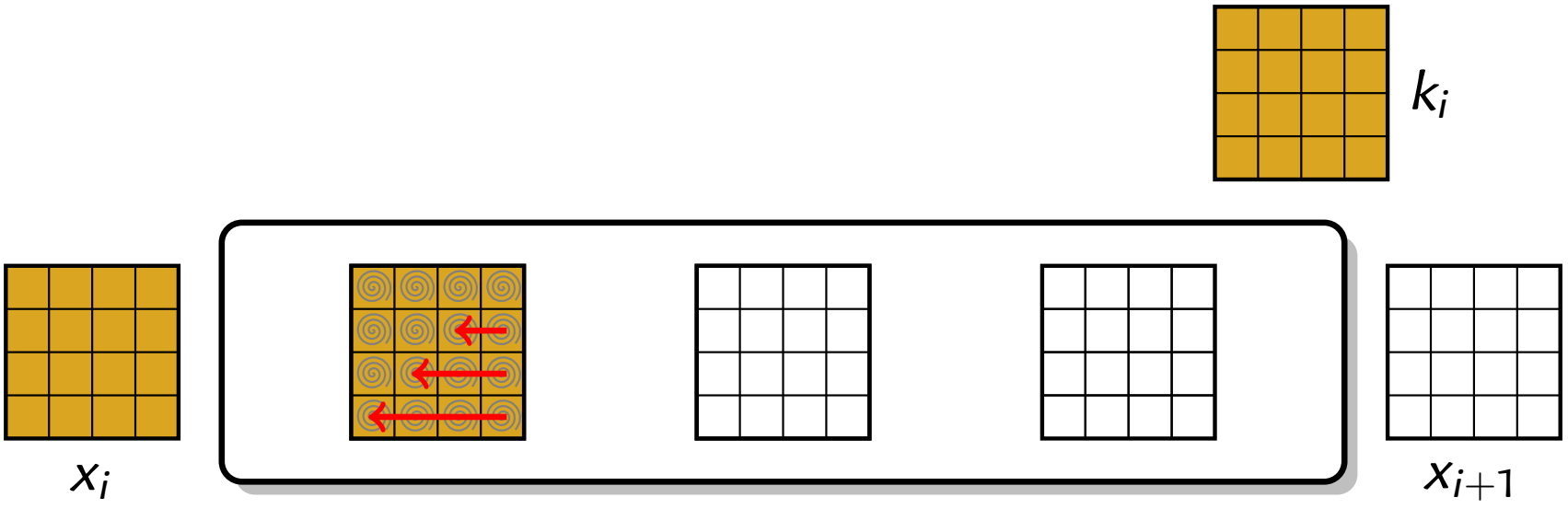
Description of the AES



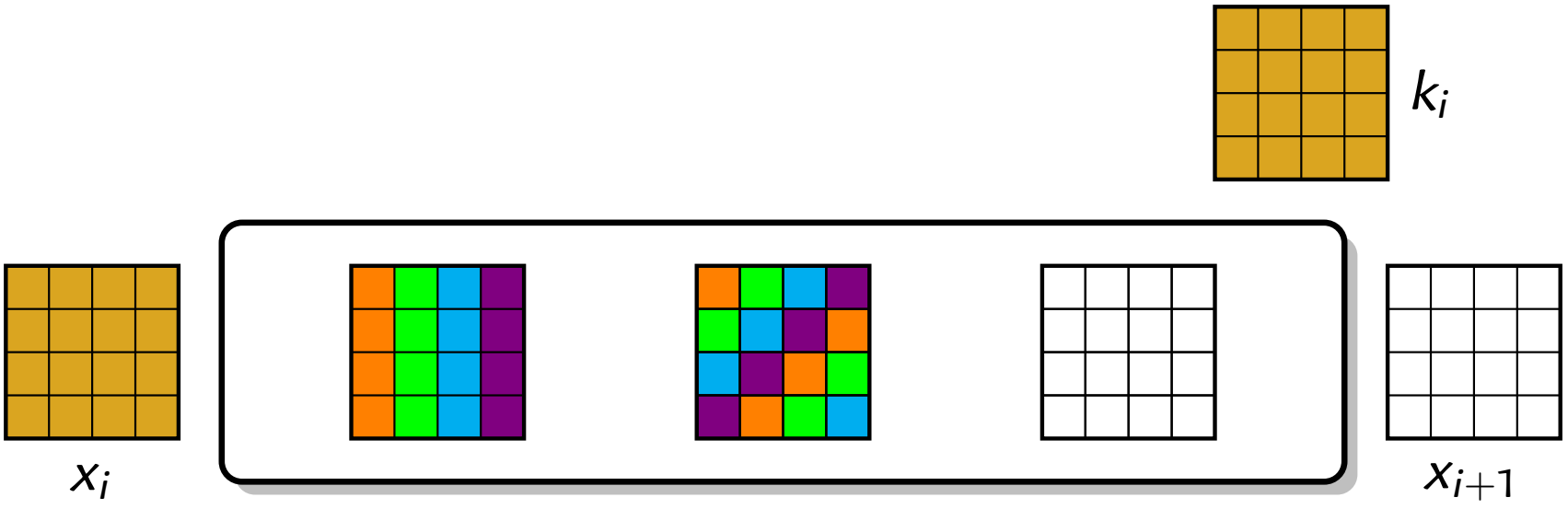
Description of the AES



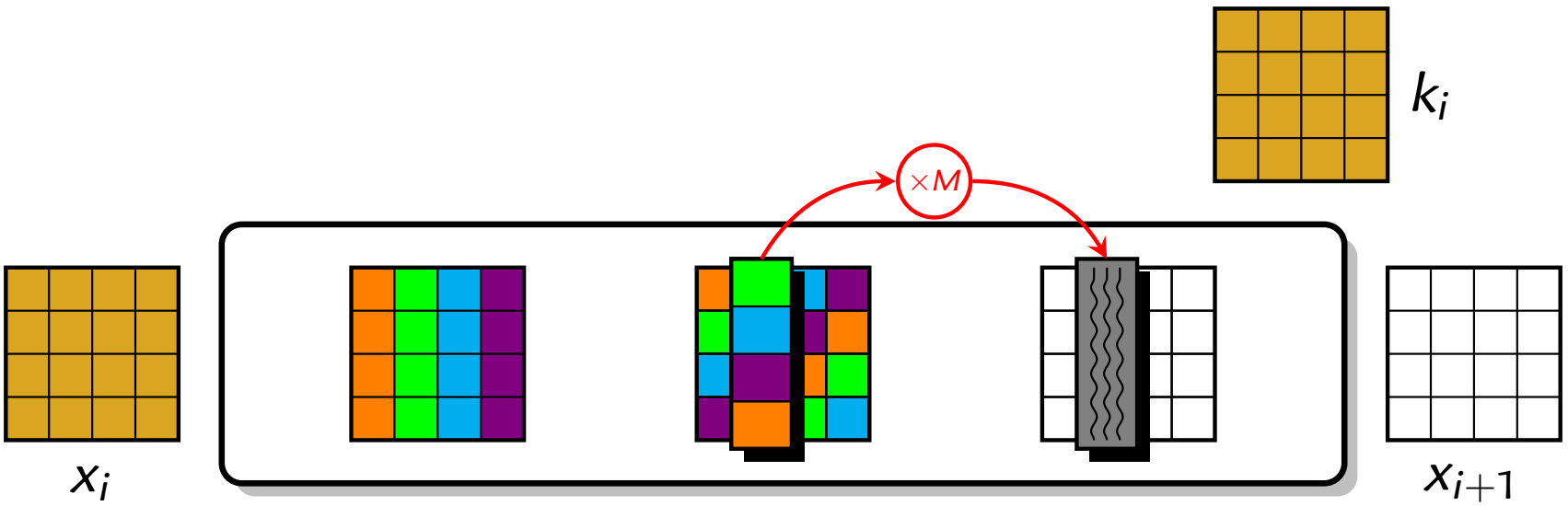
Description of the AES



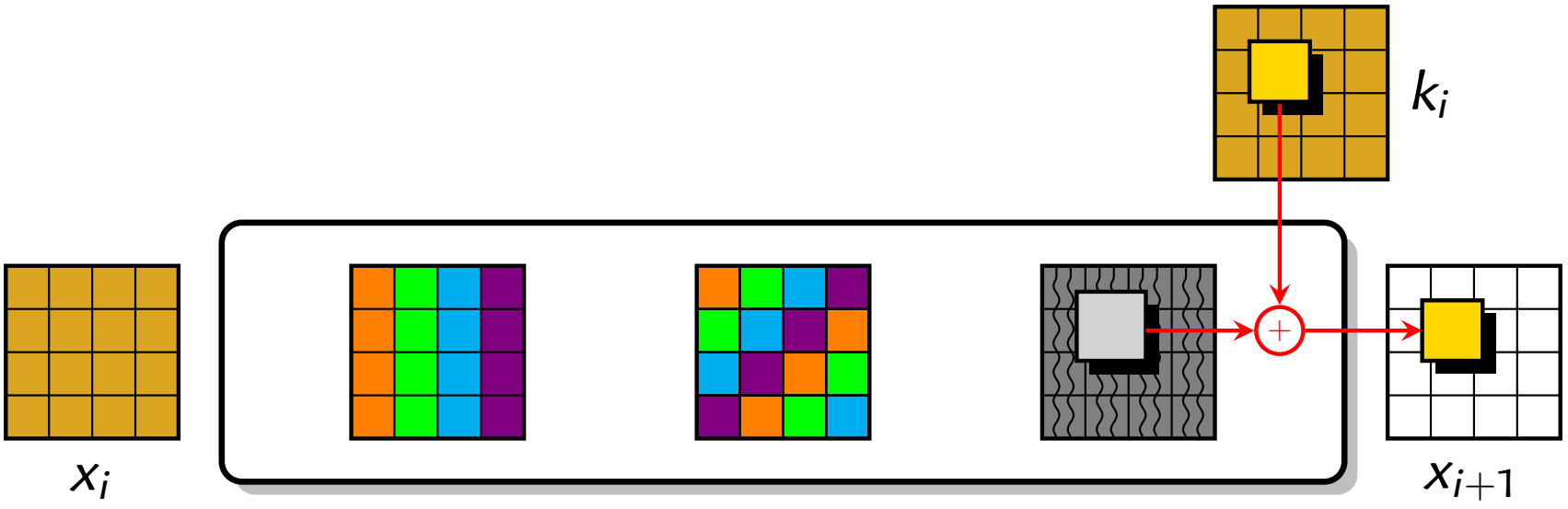
Description of the AES



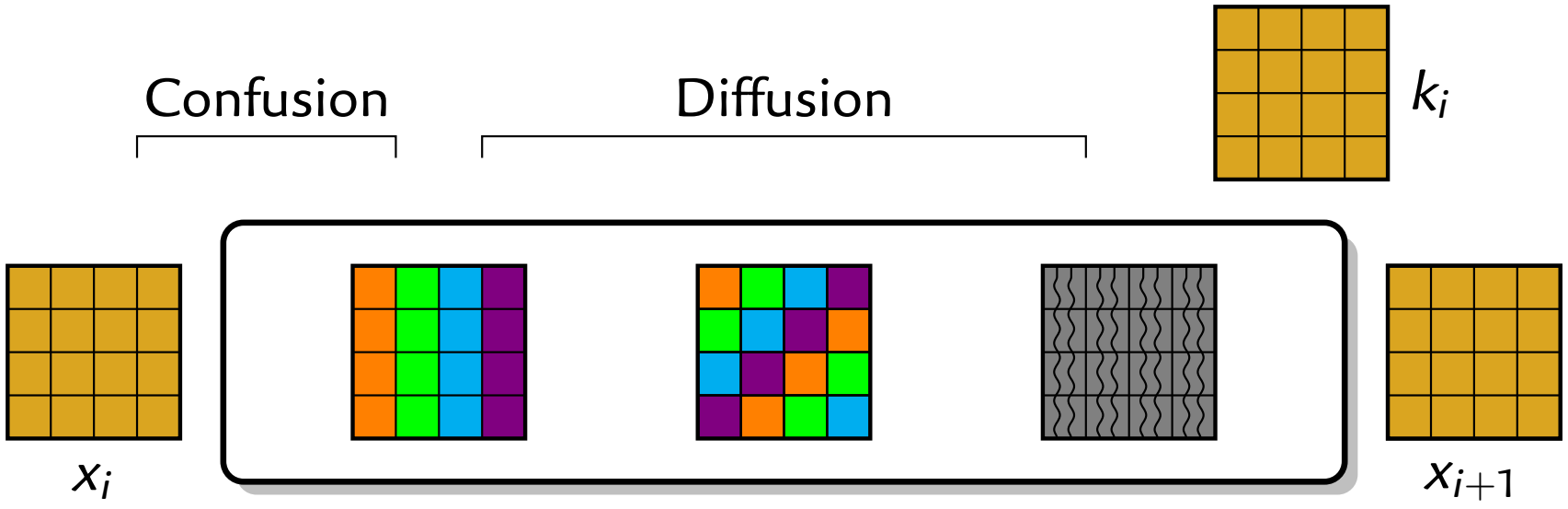
Description of the AES



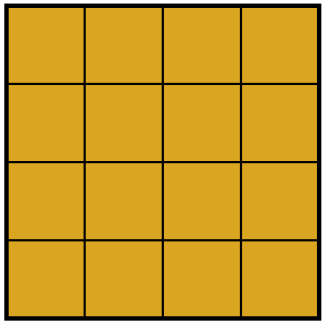
Description of the AES



Description of the AES

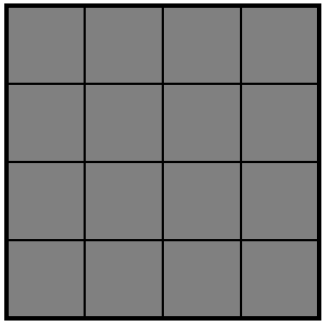


Description of the AES: the Key-Schedule

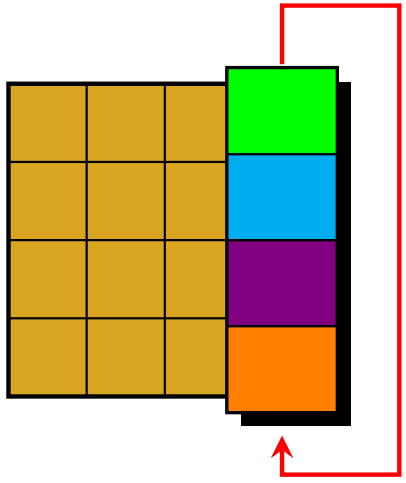


k_i

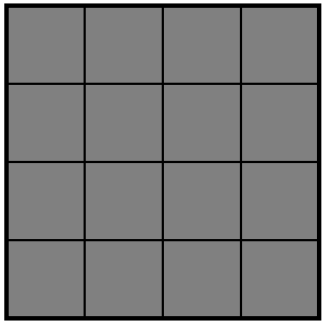
k_{i+1}



Description of the AES: the Key-Schedule

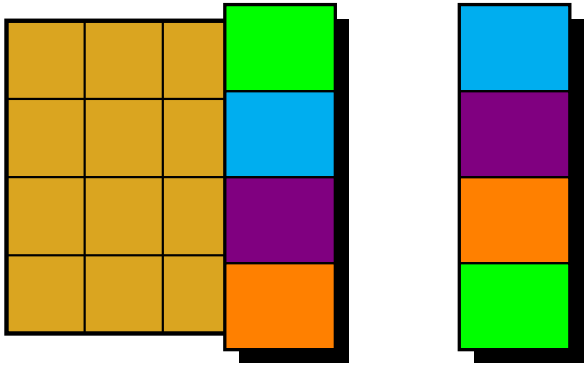


k_i

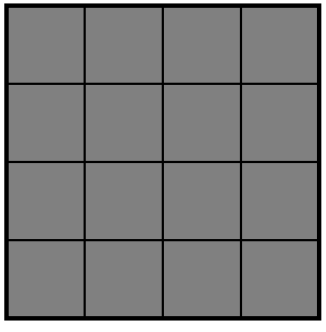


k_{i+1}

Description of the AES: the Key-Schedule

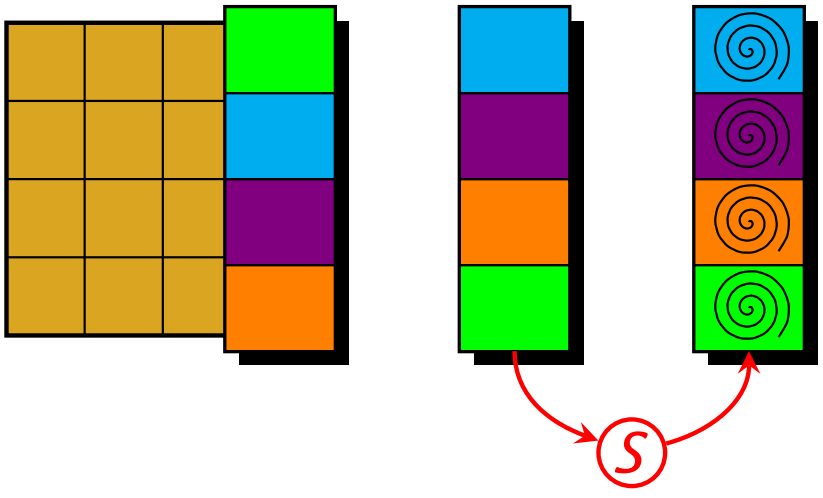


k_i

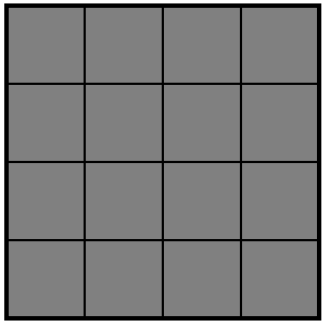


k_{i+1}

Description of the AES: the Key-Schedule

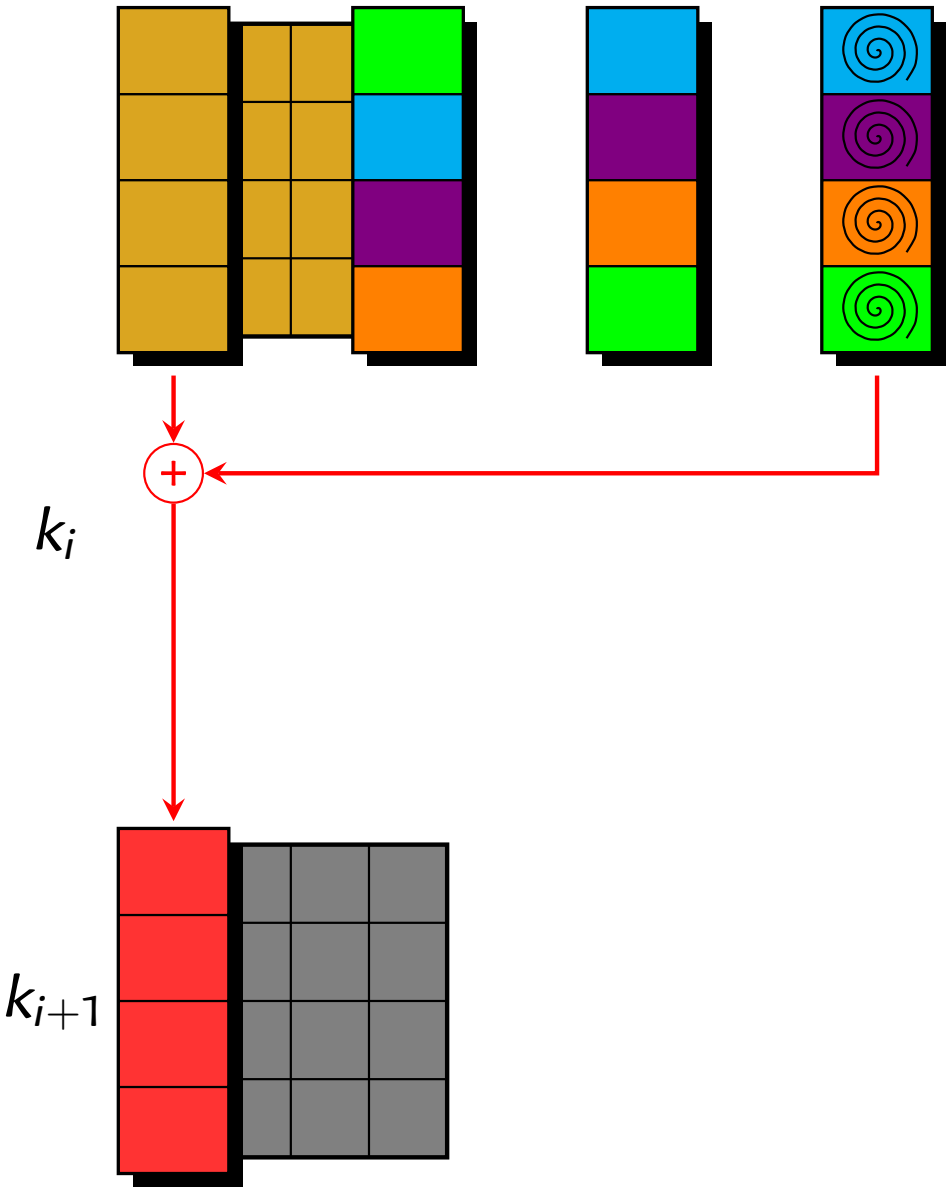


k_i

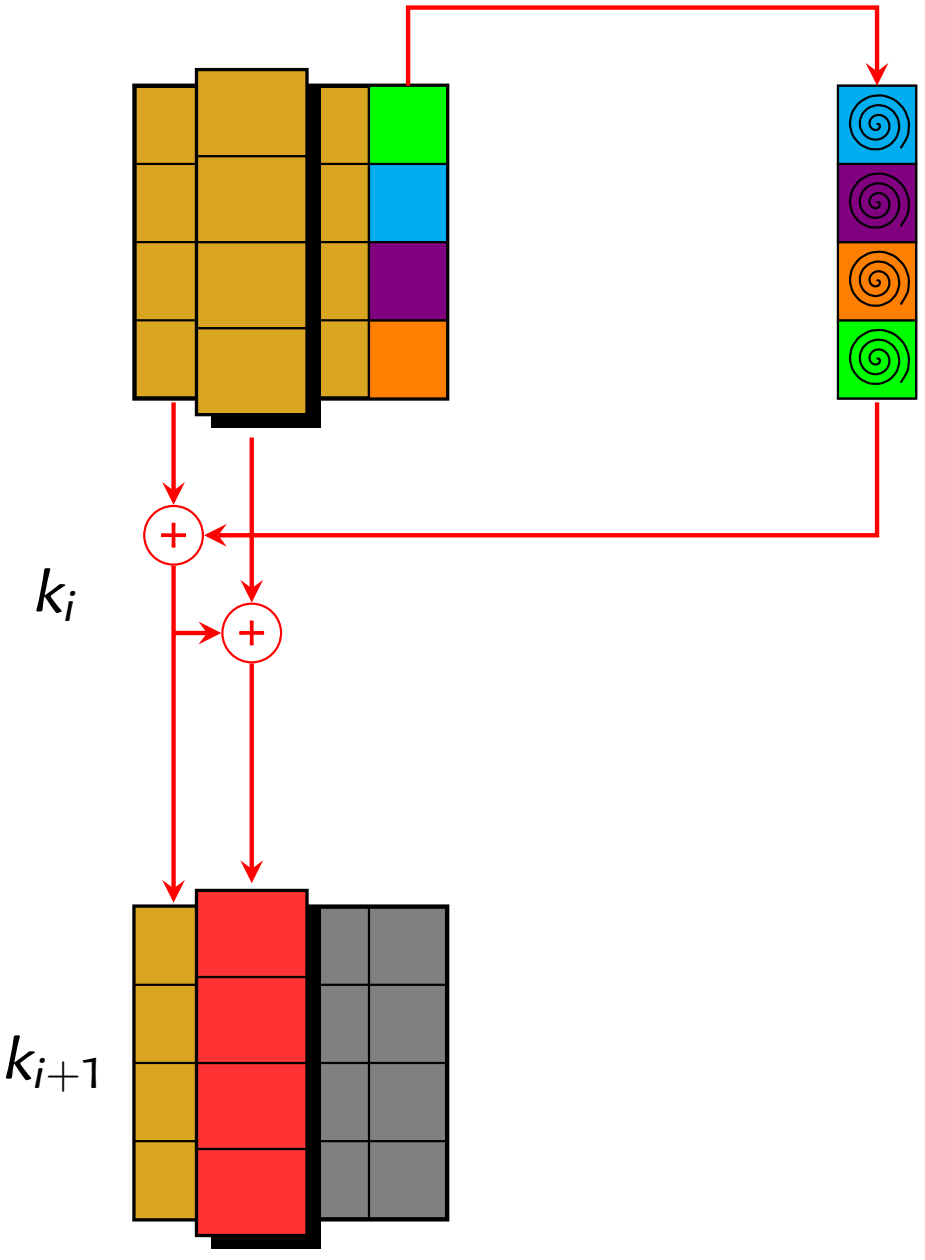


k_{i+1}

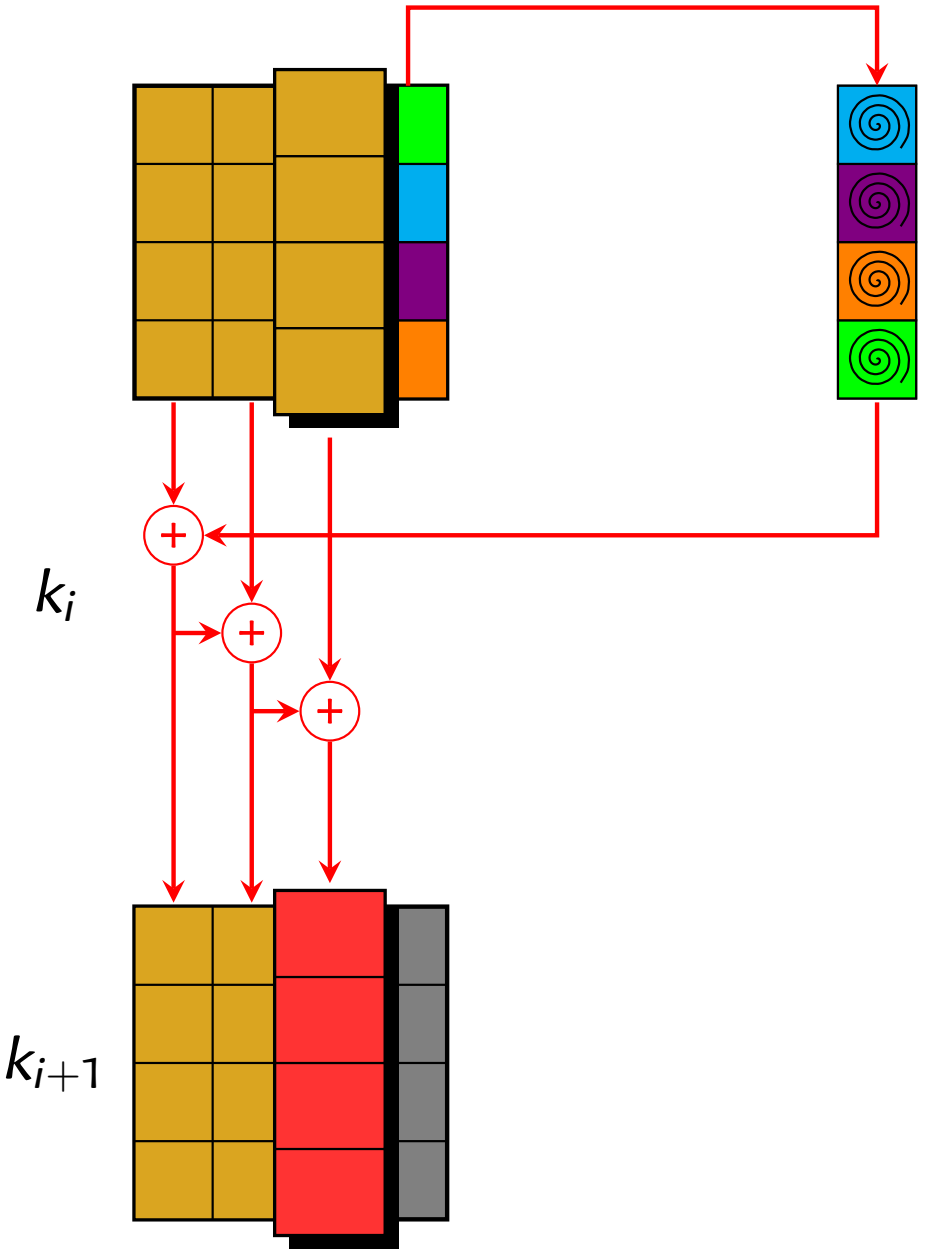
Description of the AES: the Key-Schedule



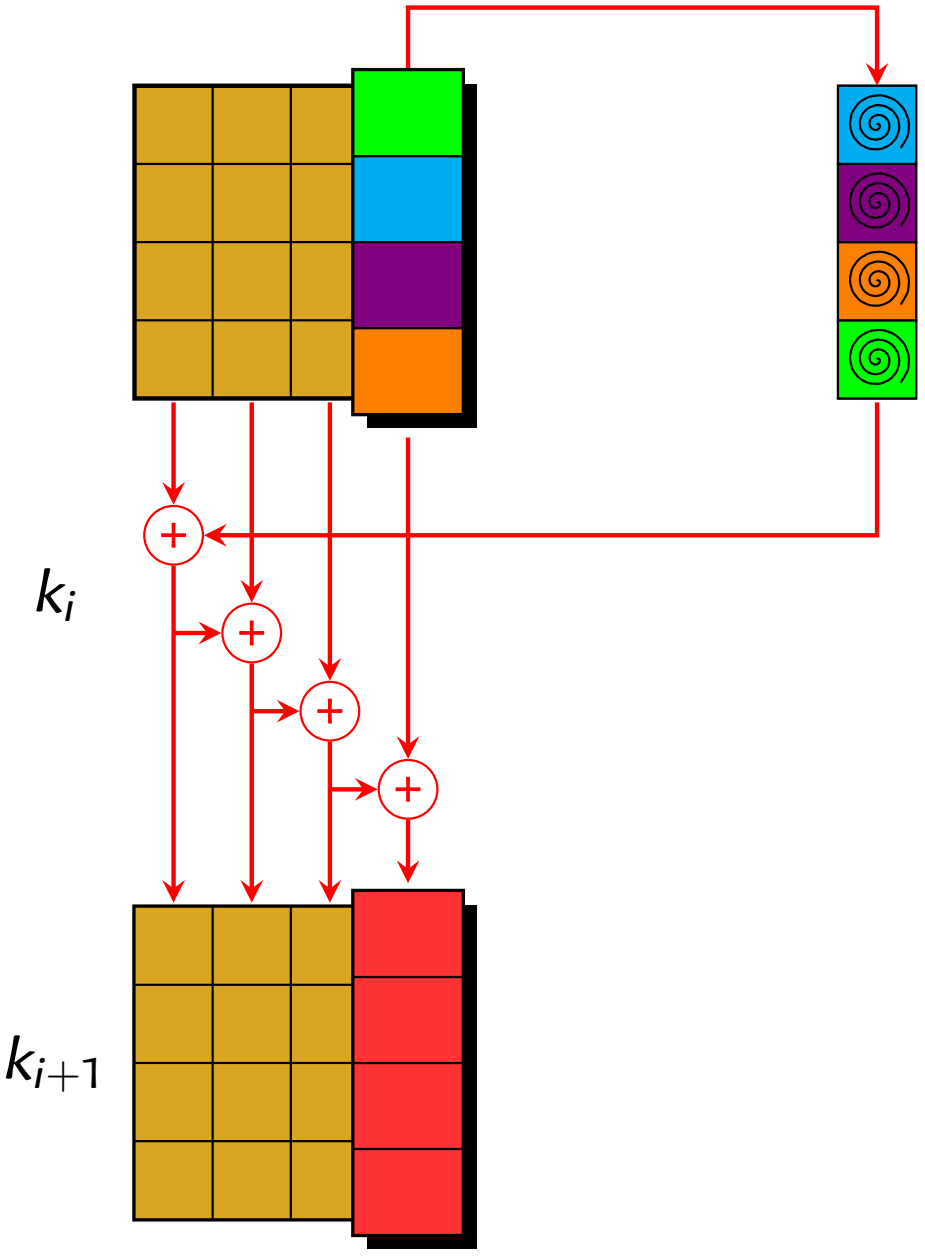
Description of the AES: the Key-Schedule



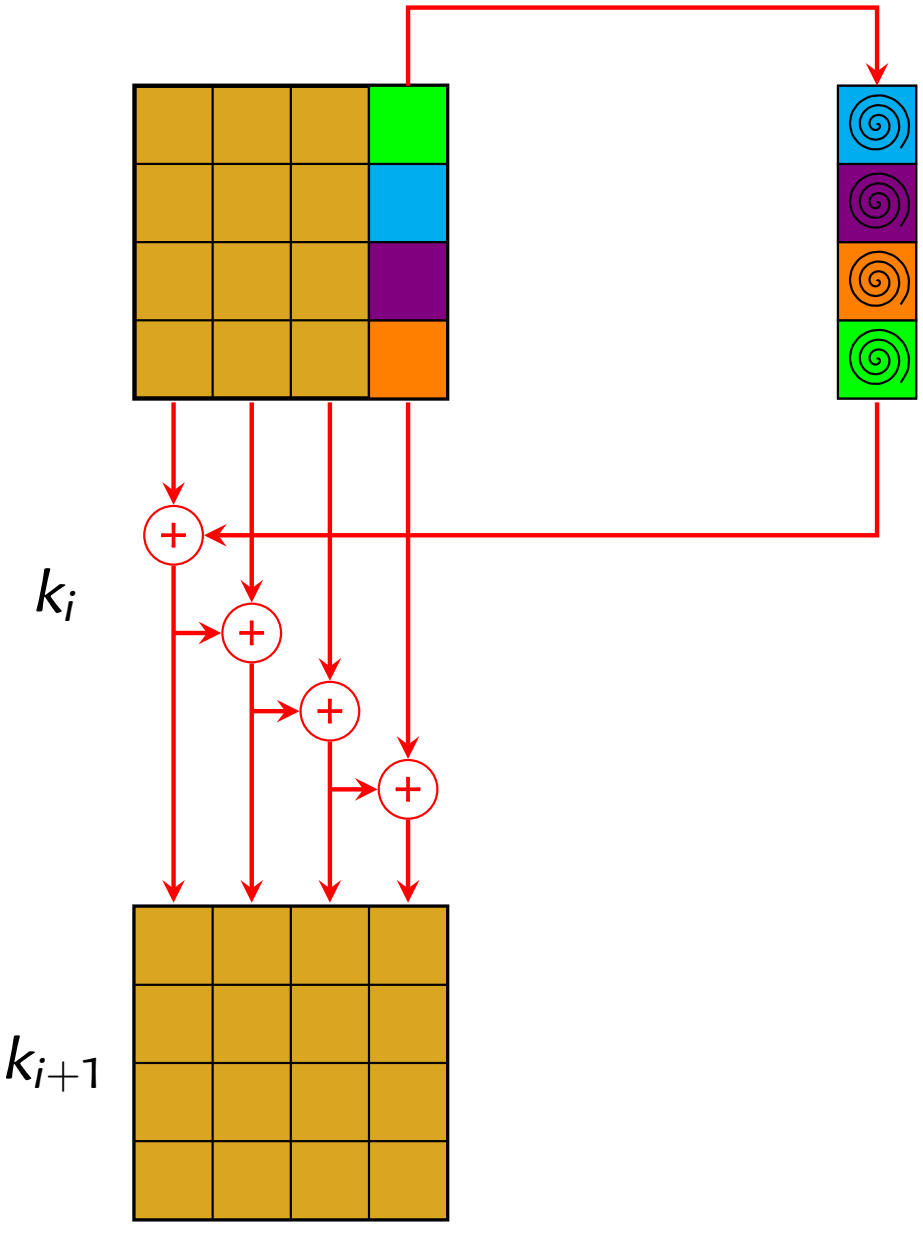
Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



Description of the AES: the Key-Schedule



Techniques for Low Data Complexity Attacks

The problem with "Usual" attack techniques

- **Statistical attacks** (*e.g.*, [impossible] differential, linear)
- **"Golden-plaintext" attacks** (*e.g.*, reflexion, slide)

They require **(VERY) LARGE QUANTITY** of data

What's Left?

- Algebraic Attacks
- Meet-in-the-Middle attacks
- Guess-and-Determine attacks

Techniques for Low Data Complexity Attacks

The problem with "Usual" attack techniques

- **Statistical attacks** (*e.g.*, [impossible] differential, linear)
- **"Golden-plaintext" attacks** (*e.g.*, reflexion, slide)

They require **(VERY) LARGE QUANTITY** of data

What's Left?

- Algebraic Attacks
- Meet-in-the-Middle attacks
- Guess-and-Determine attacks

The AES Has a Clean Description over $\mathbb{F}_{256}$

$$x_0[j] = P[j] + k_0[j]$$
$$y_i[j] = S(x_i[j])$$
$$x_{i+1} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} y_i[0] & y_i[4] & y_i[8] & y_i[12] \\ y_i[5] & y_i[9] & y_i[13] & y_i[1] \\ y_i[10] & y_i[14] & y_i[2] & y_i[6] \\ y_i[15] & y_i[3] & y_i[7] & y_i[11] \end{pmatrix} + k_{i+1}$$

- **Equation** = linear combination of **Terms** over $\mathbb{F}_{256}$
- **Term** = X_i or $S(X_i)$

The equations are:

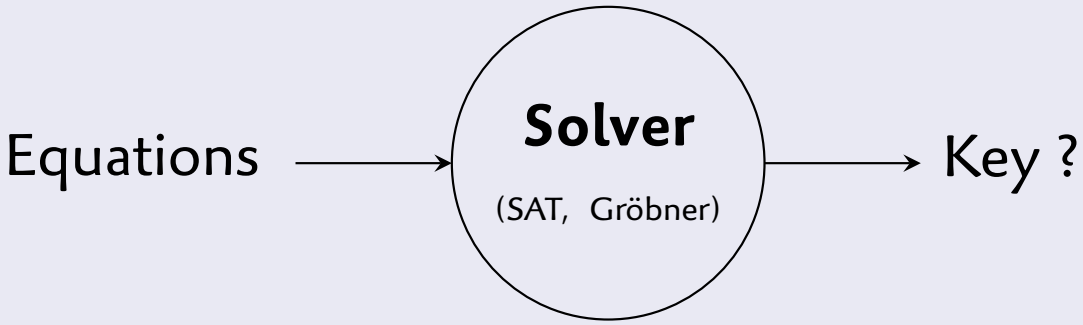
- sparse: each equation relates, at most, five variables
- structured: each variable appears in, at most, four equations
  - A **Gröbner basis** of the equations is **known** (but useless)

Working With the Equations

Is it a Problem?

- **Concerns** about the AES's algebraic simplicity have been expressed several times

Algebraic Cryptanalysis: the Direct Approach



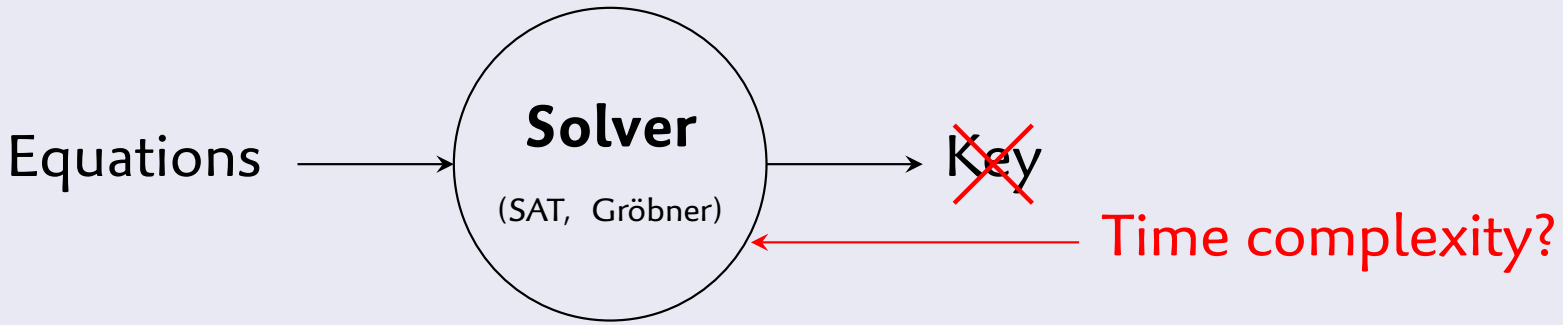
- 2002 : Claims that a Gröbner-like solver **breaks** AES
- 2005 : Previous claim **debunked**
  - No practical results
  - No interesting upper-bound on solving time

Working With the Equations

Is it a Problem?

- **Concerns** about the AES's algebraic simplicity have been expressed several times

Algebraic Cryptanalysis: the Direct Approach



- 2002 : Claims that a Gröbner-like solver ☠ **breaks** ☠ AES
- 2005 : Previous claim **debunked**
  - No practical results
  - No interesting upper-bound on solving time

The Problem With Algebraic Attacks

Why is it failing?

- Main obstacle = S-box
- has only "bad" representations

Pluging the description of the S-box into the equations makes them hard.

$\Longrightarrow$ Could we not do that?

Goal

Find a way to solve the equations **independently of a particular choice of** S

Paradigm Shift

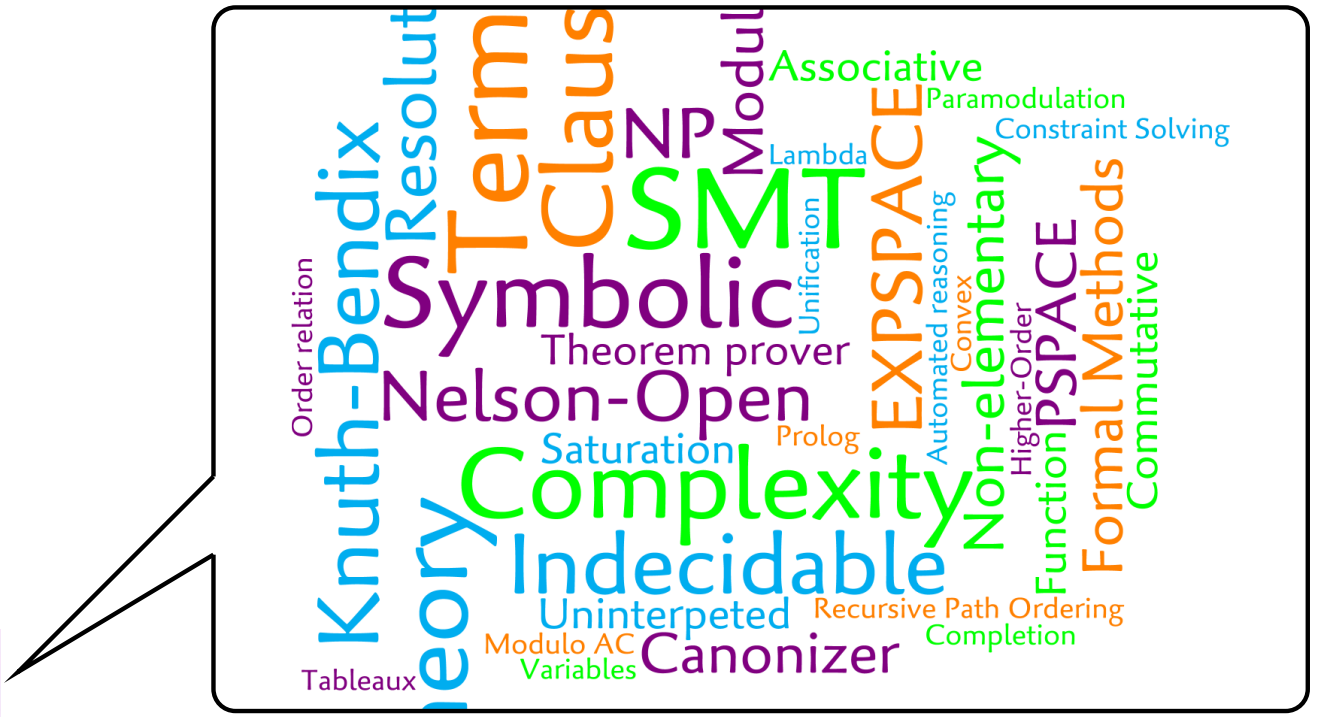
Old Goal

Break Cryptographic Stuff

Paradigm Shift

New Goal

(efficiently) Solve linear equations over $\mathbb{F}_{256}$ with an **uninterpreted** permutation symbol



Paradigm Shift

New Goal

(efficiently) Solve linear equations over $\mathbb{F}_{256}$ with an **uninterpreted** permutation symbol

DISCLAIMER: there is (probably) a lot of existing litterature devoted to this particular problem, or to similar-looking ones, that I am not aware of.

In this talk, I present our own answers, which are based on cryptographic techniques.

They crucially rely on the fact that variables live in a finite domain.

Techniques for Low Data Complexity Attacks

The problem with "Usual" attack techniques

- **Statistical attacks** (*e.g.*, [impossible] differential, linear)
- **"Golden-plaintext" attacks** (*e.g.*, reflexion, slide)

They require **(VERY) LARGE QUANTITY** of data

What's Left?

- ~~Algebraic Attacks~~
- Meet-in-the-Middle attacks
- Guess-and-Determine attacks

Meet-in-the-Middle Attacks

A Well-Known Example: Double-DES

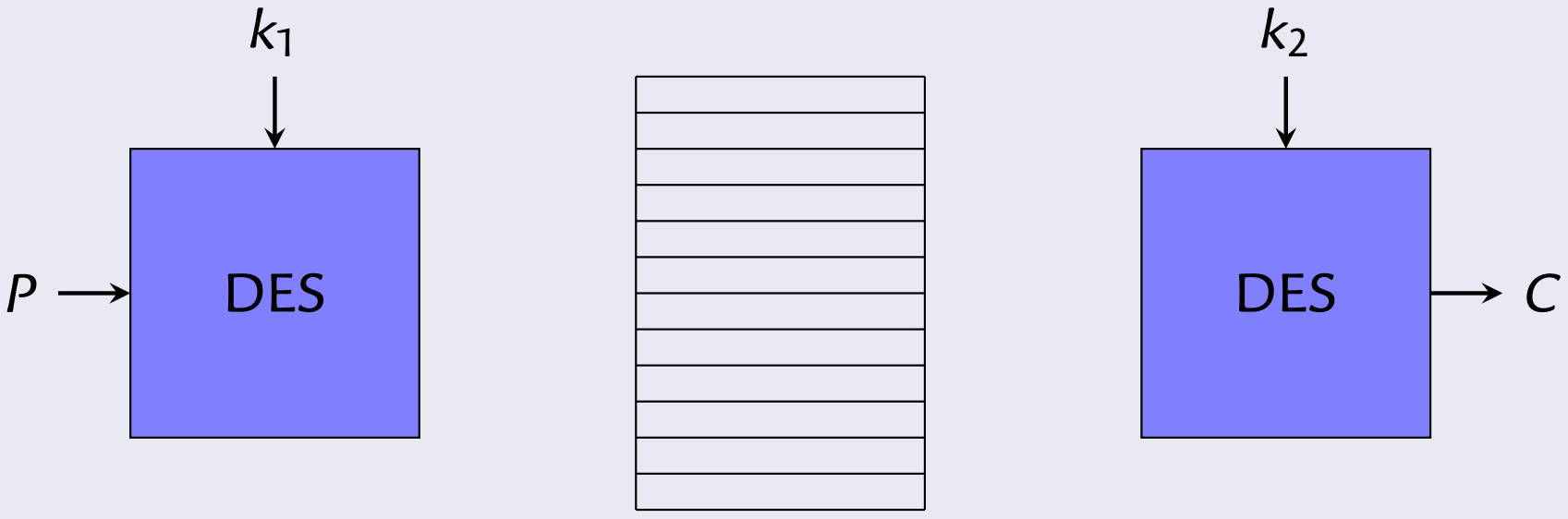
$$2DES_{k_1,k_2} = DES_{k_2} \circ DES_{k_1}$$

- Initialize a Hash Table
- For all k_1, store $M = DES_{k_1}(P) \rightarrow k_1$
- For all k_2, look-up $M = DES^{-1}_{k_2}(C)$

Time complexity $\approx 2^k$ encryptions, with $2k$-bit keys!

Meet-in-the-Middle Attacks

A Well-Known Example: Double-DES



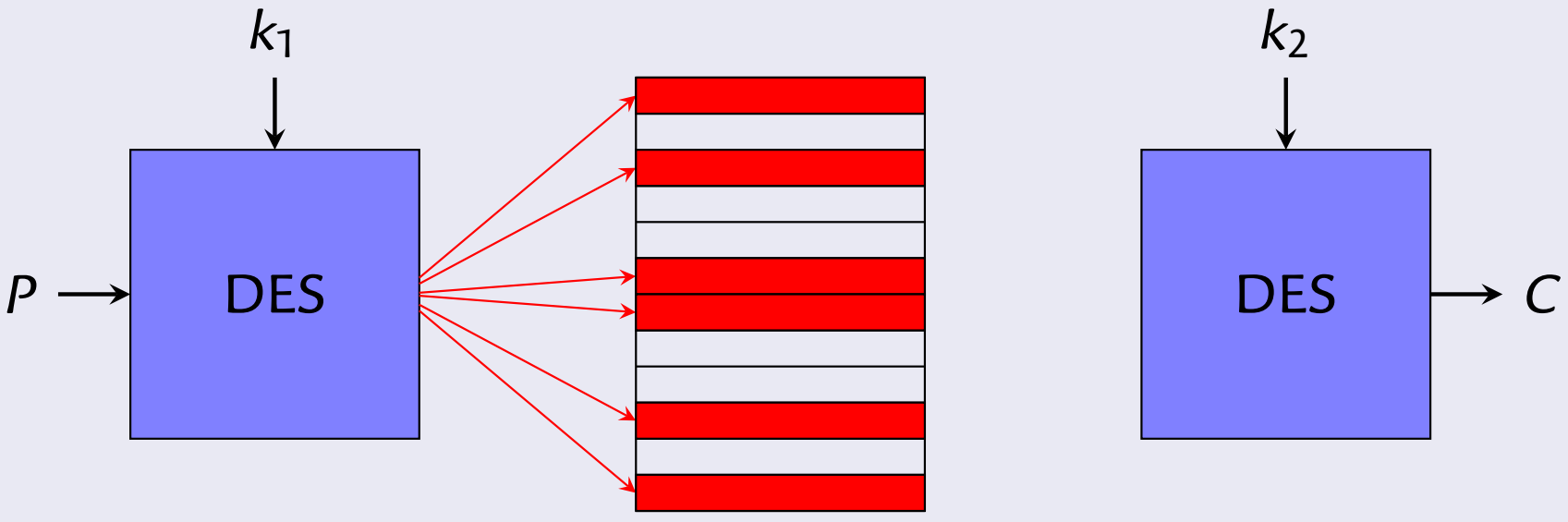
$$2DES_{k_1,k_2} = DES_{k_2} \circ DES_{k_1}$$

- Initialize a Hash Table
- For all k_1, store $M = DES_{k_1}(P) \rightarrow k_1$
- For all k_2, look-up $M = DES_{k_2}^{-1}(C)$

Time complexity $\approx 2^k$ encryptions, with $2k$-bit keys!

Meet-in-the-Middle Attacks

A Well-Known Example: Double-DES



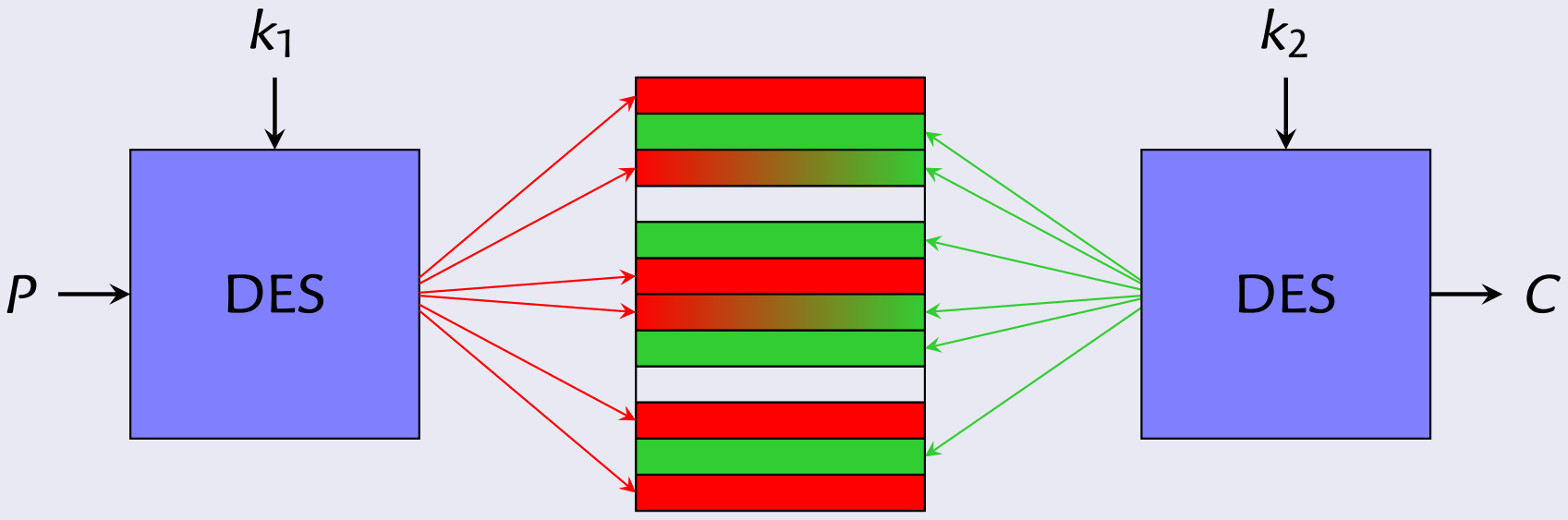
$$2DES_{k_1,k_2} = DES_{k_2} \circ DES_{k_1}$$

- Initialize a Hash Table
- For all k_1, store $M = DES_{k_1}(P) \rightarrow k_1$
- For all k_2, look-up $M = DES^{-1}_{k_2}(C)$

Time complexity $\approx 2^k$ encryptions, with $2k$-bit keys!

Meet-in-the-Middle Attacks

A Well-Known Example: Double-DES



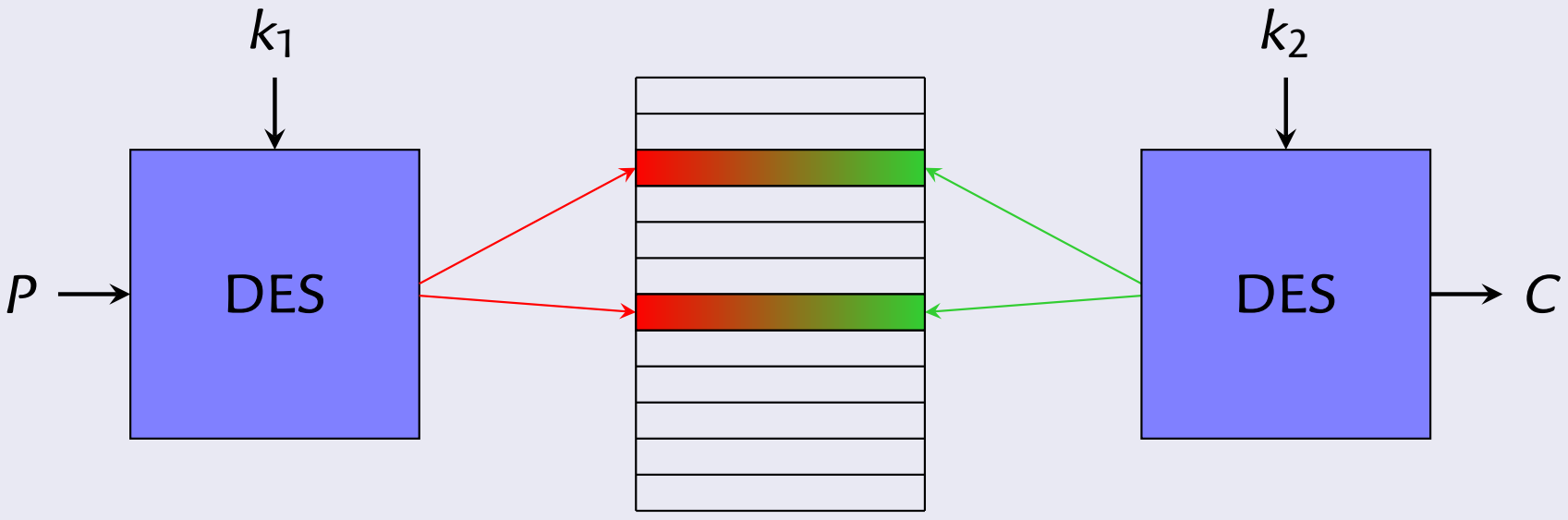
$$2DES_{k_1,k_2} = DES_{k_2} \circ DES_{k_1}$$

- Initialize a Hash Table
- For all k_1, store $M = DES_{k_1}(P) \rightarrow k_1$
- For all k_2, look-up $M = DES^{-1}_{k_2}(C)$

Time complexity $\approx 2^k$ encryptions, with $2k$-bit keys!

Meet-in-the-Middle Attacks

A Well-Known Example: Double-DES



$$2DES_{k_1,k_2} = DES_{k_2} \circ DES_{k_1}$$

- Initialize a Hash Table
- For all k_1, store $M = DES_{k_1}(P) \rightarrow k_1$
- For all k_2, look-up $M = DES^{-1}_{k_2}(C)$

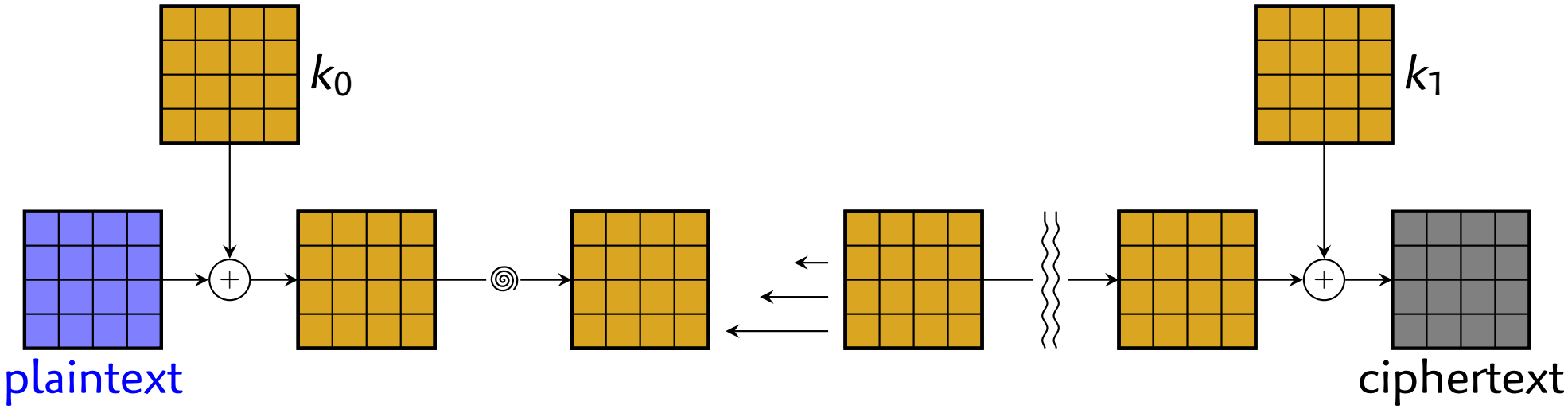
Time complexity $\approx 2^k$ encryptions, with $2k$-bit keys!

Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

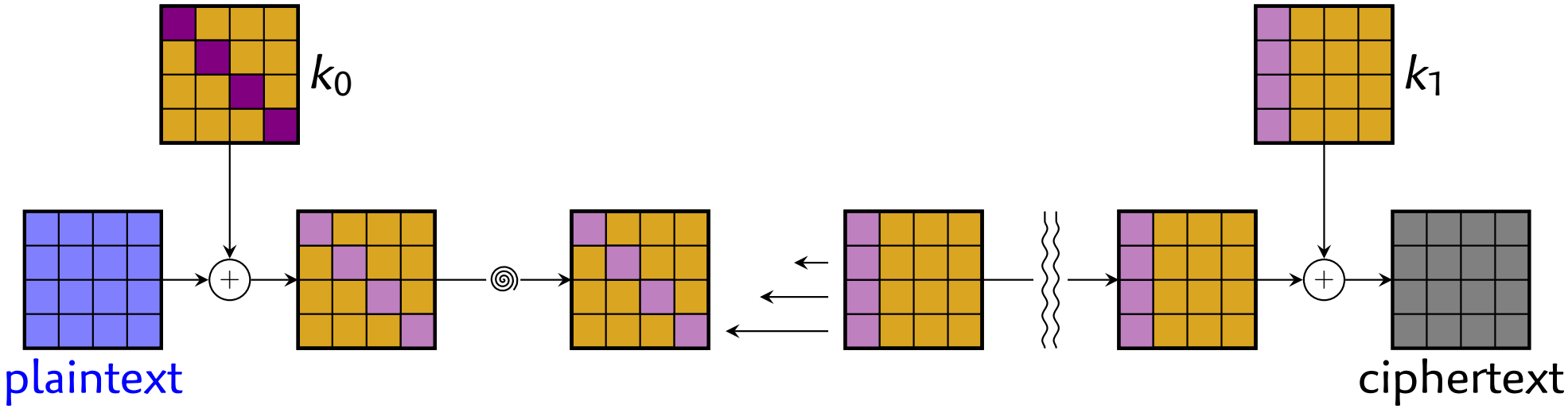


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

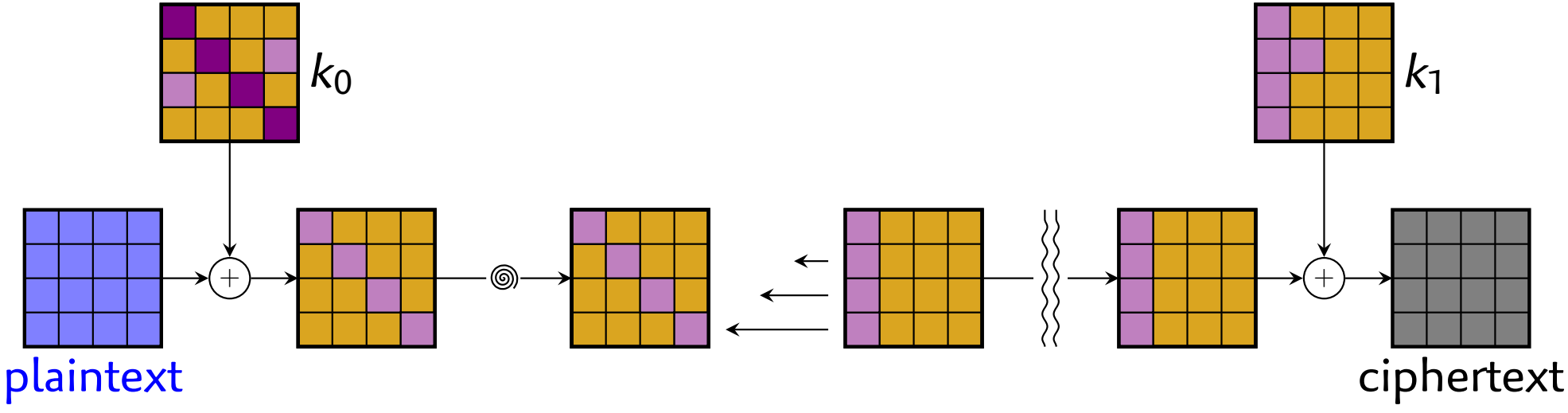


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

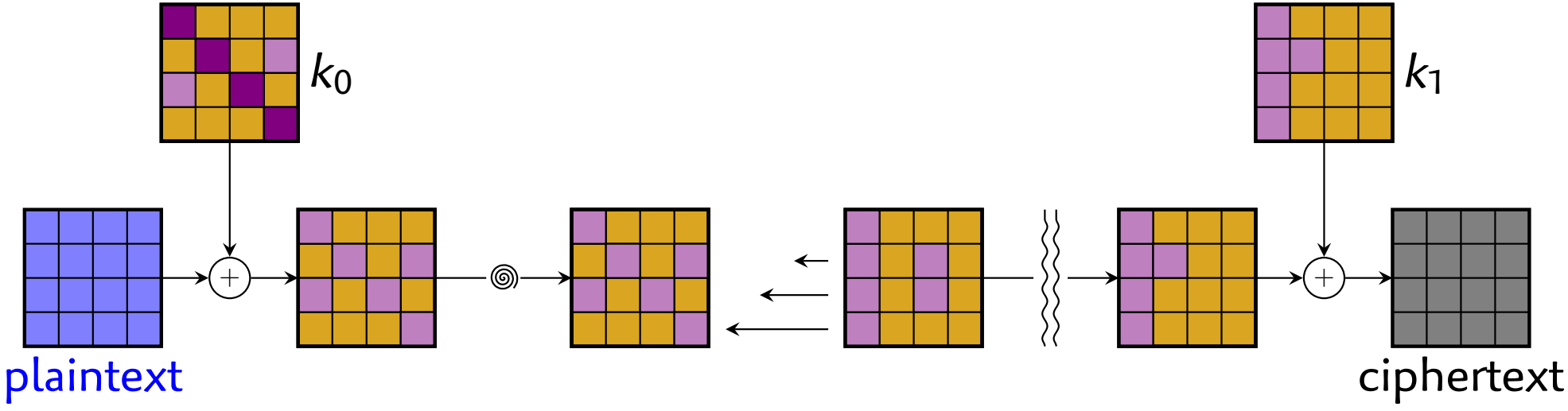


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

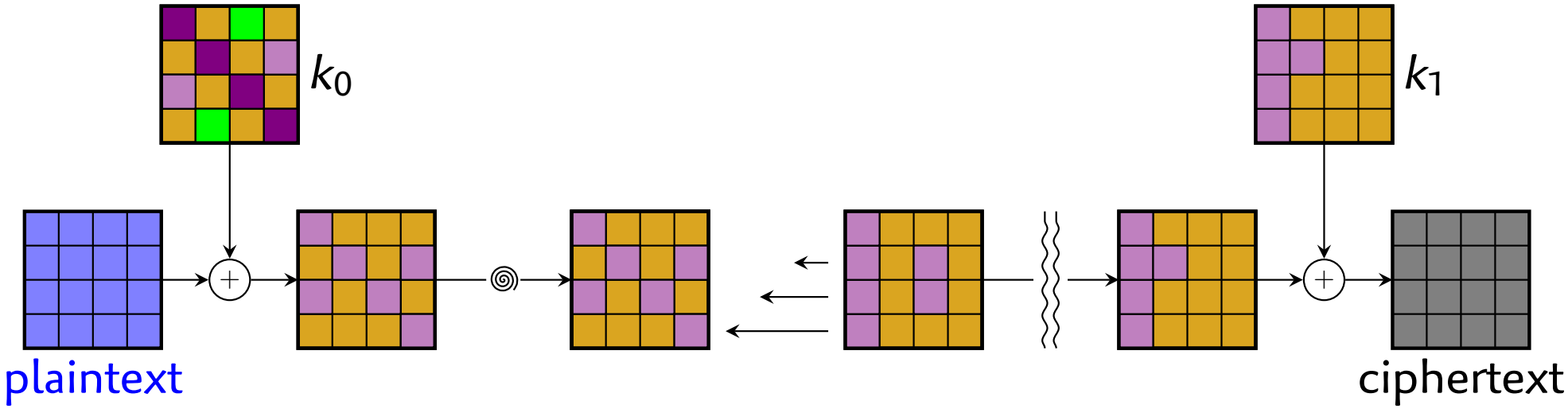


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

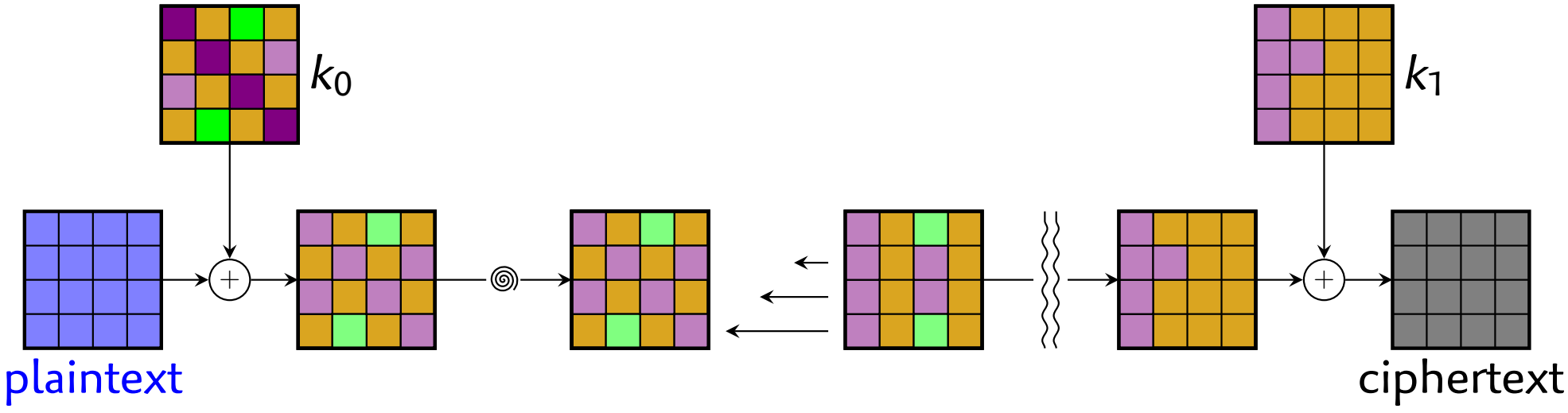


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

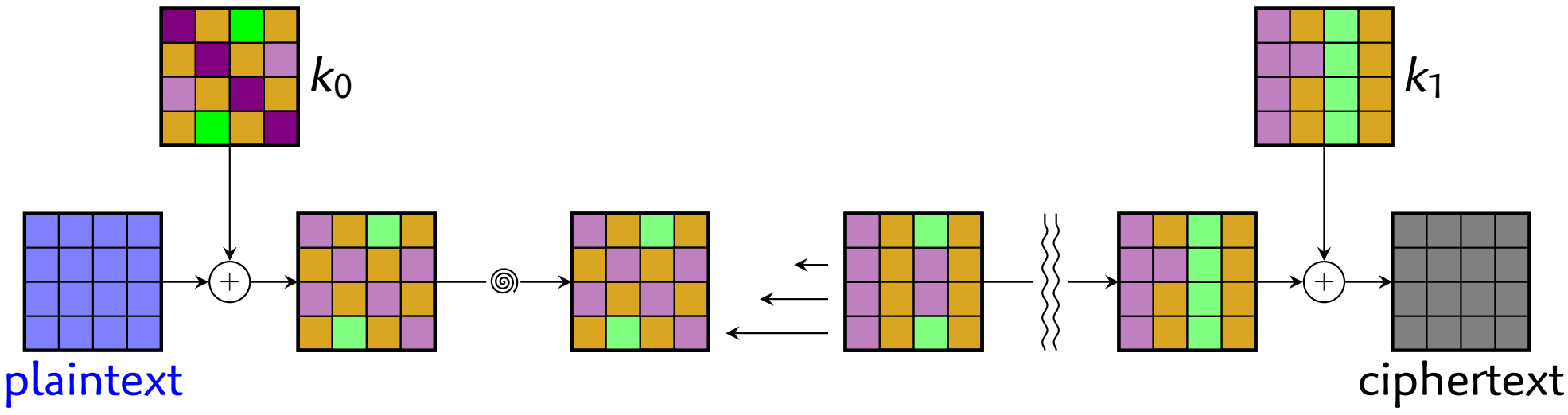


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

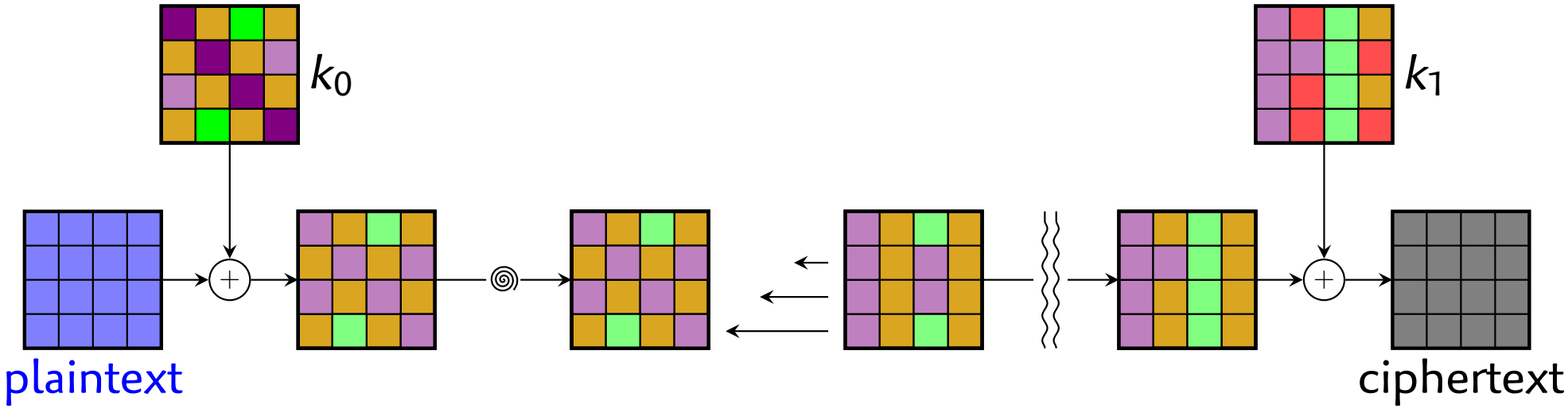


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

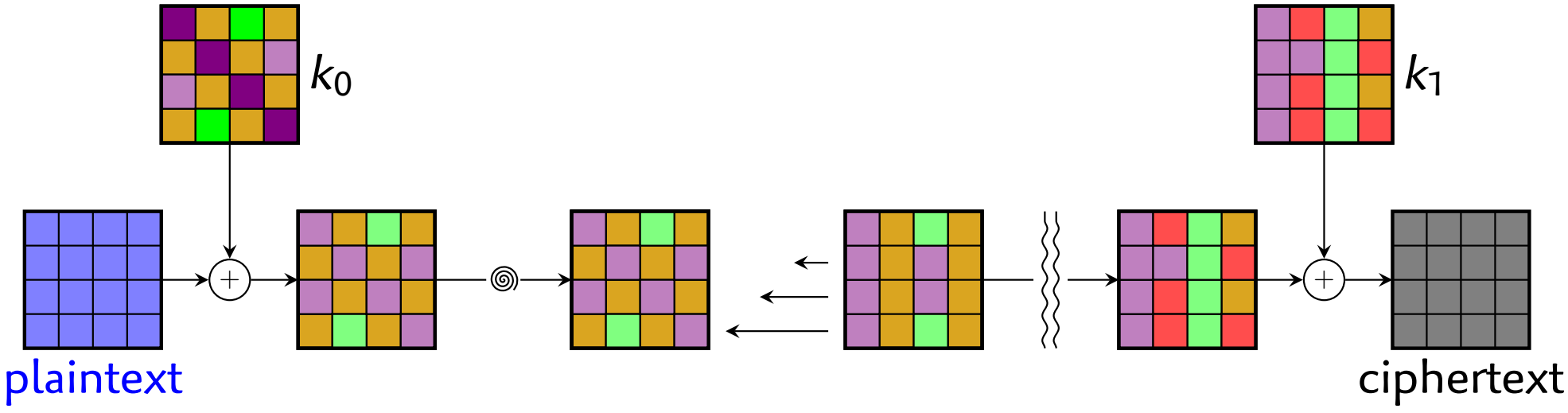


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

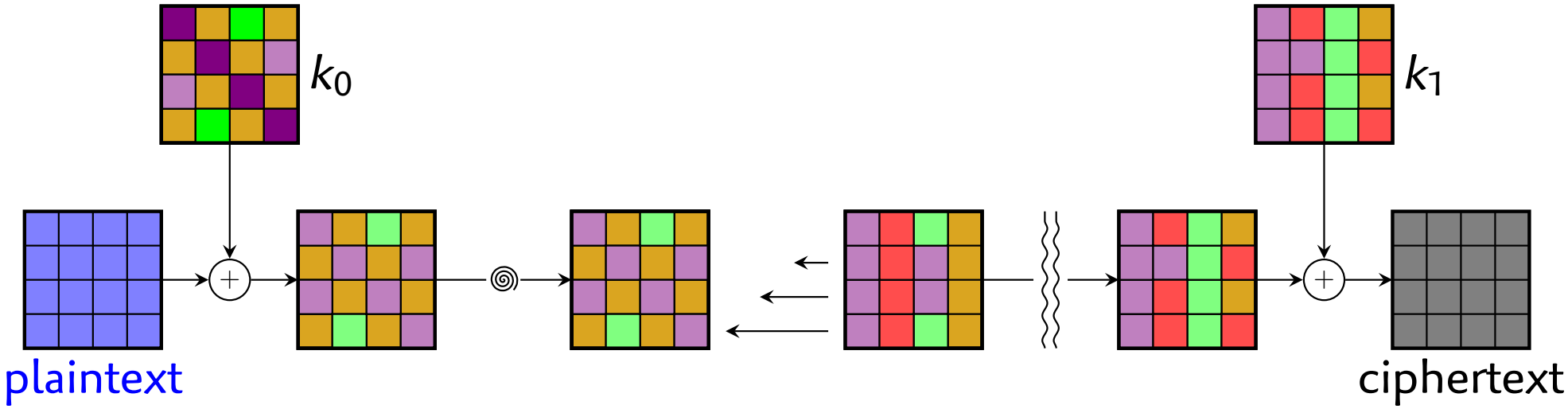


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

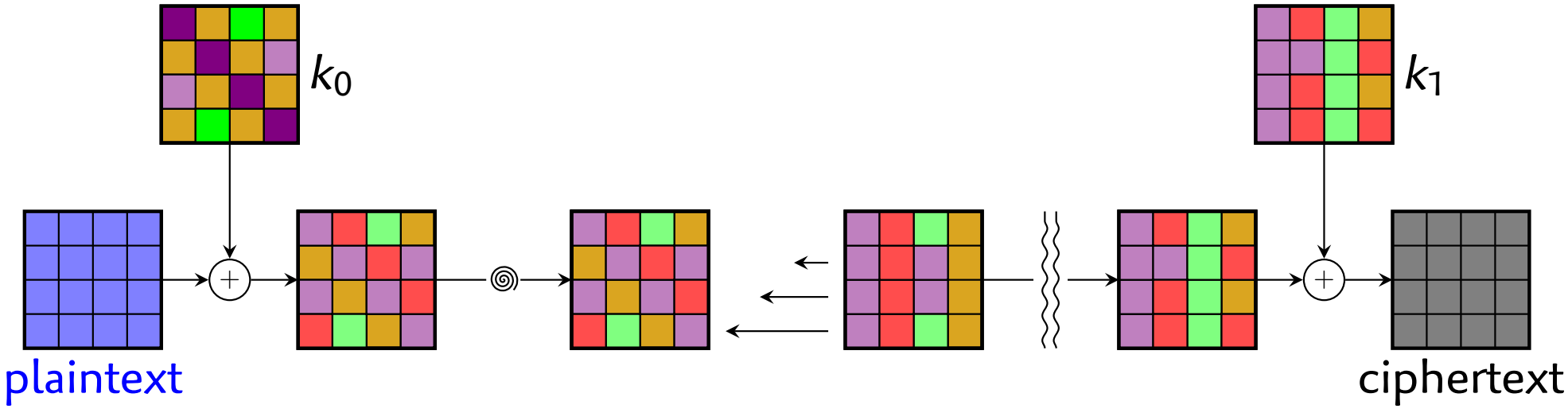


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

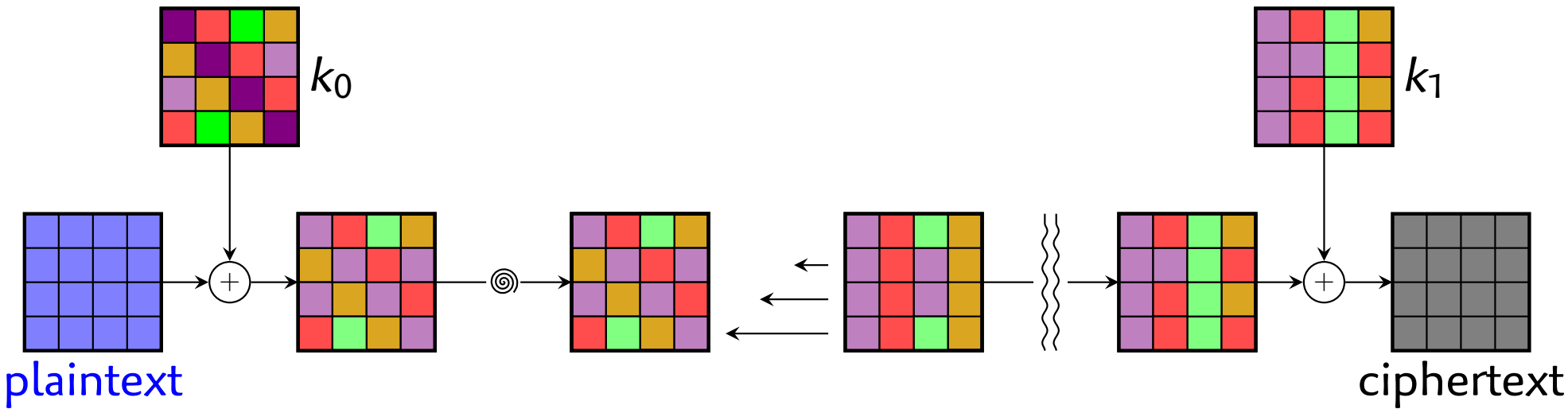


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

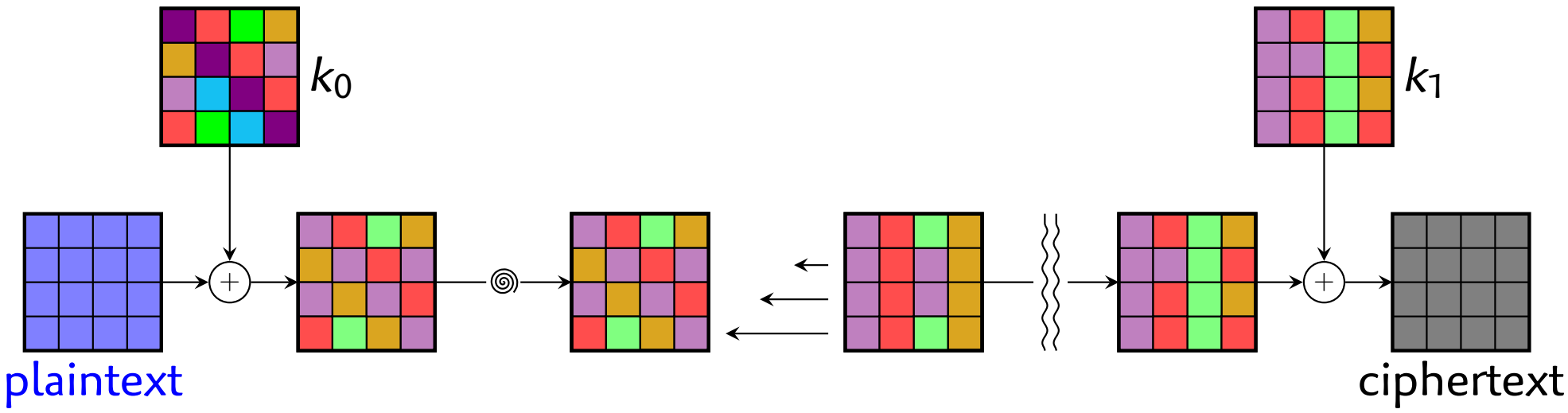


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

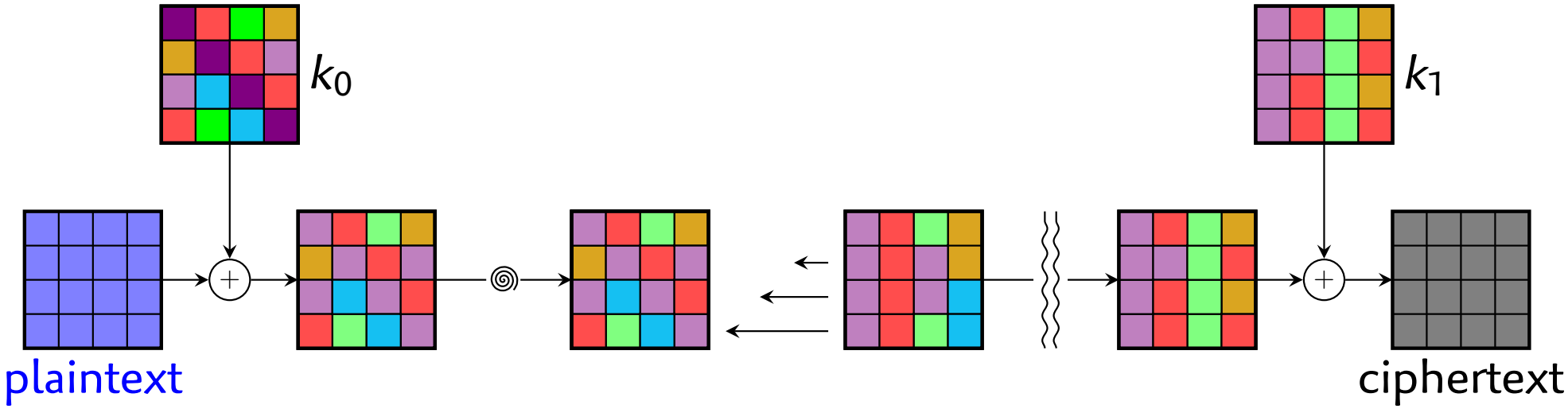


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

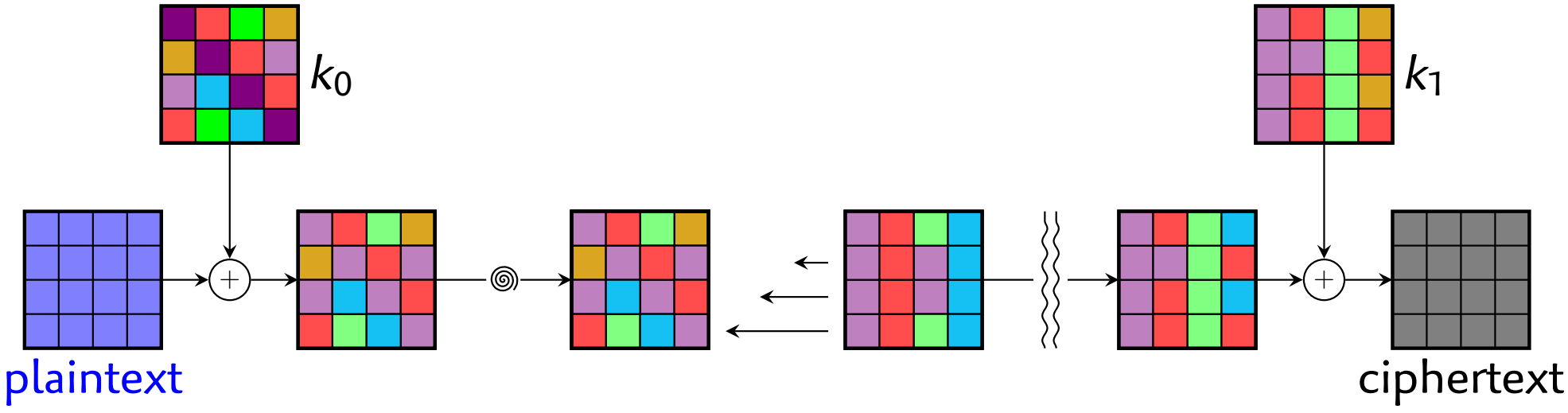


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

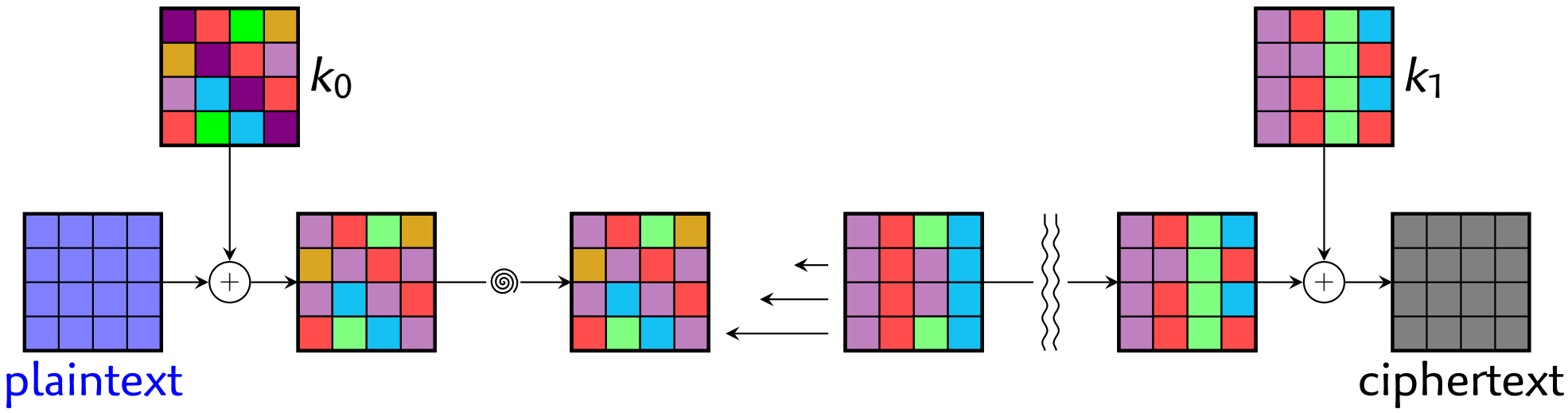


Guess-and-Determine Attacks on the AES

The Big Picture:

1. **Guess** the value of some variables
   - Complexity is exponential in # guessed variables
2. **Determine** the values of the others w/ quick computation
3. **Check** all the equations

A single AES round, a single Plaintext/Ciphertext pair

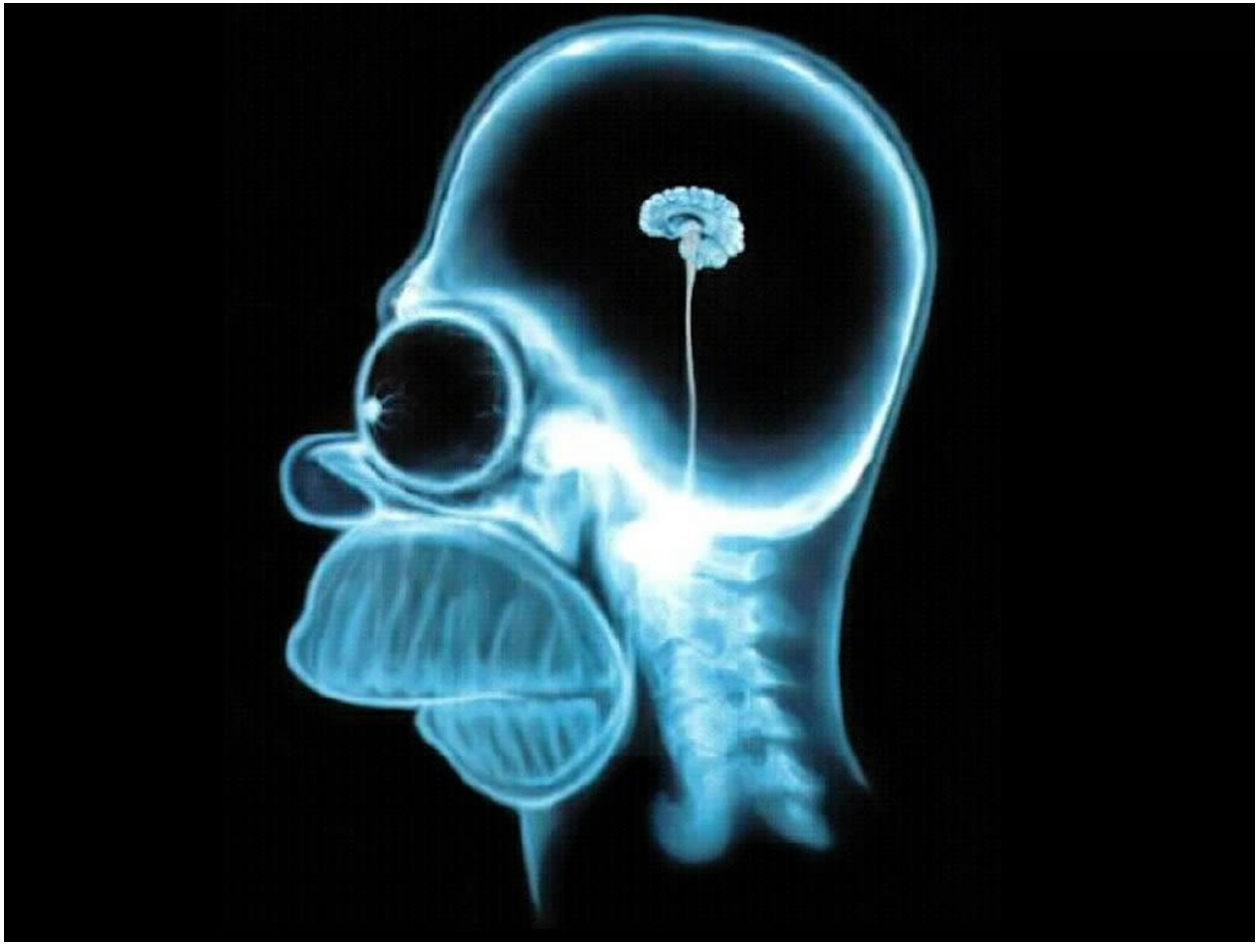


Simple in Theory...

- After a few Hours of doing this by hand
- larger systems (more rounds) → it gets even harder

Simple in Theory...

- After a few Days of doing this by hand
- larger systems (more rounds) → it gets even harder



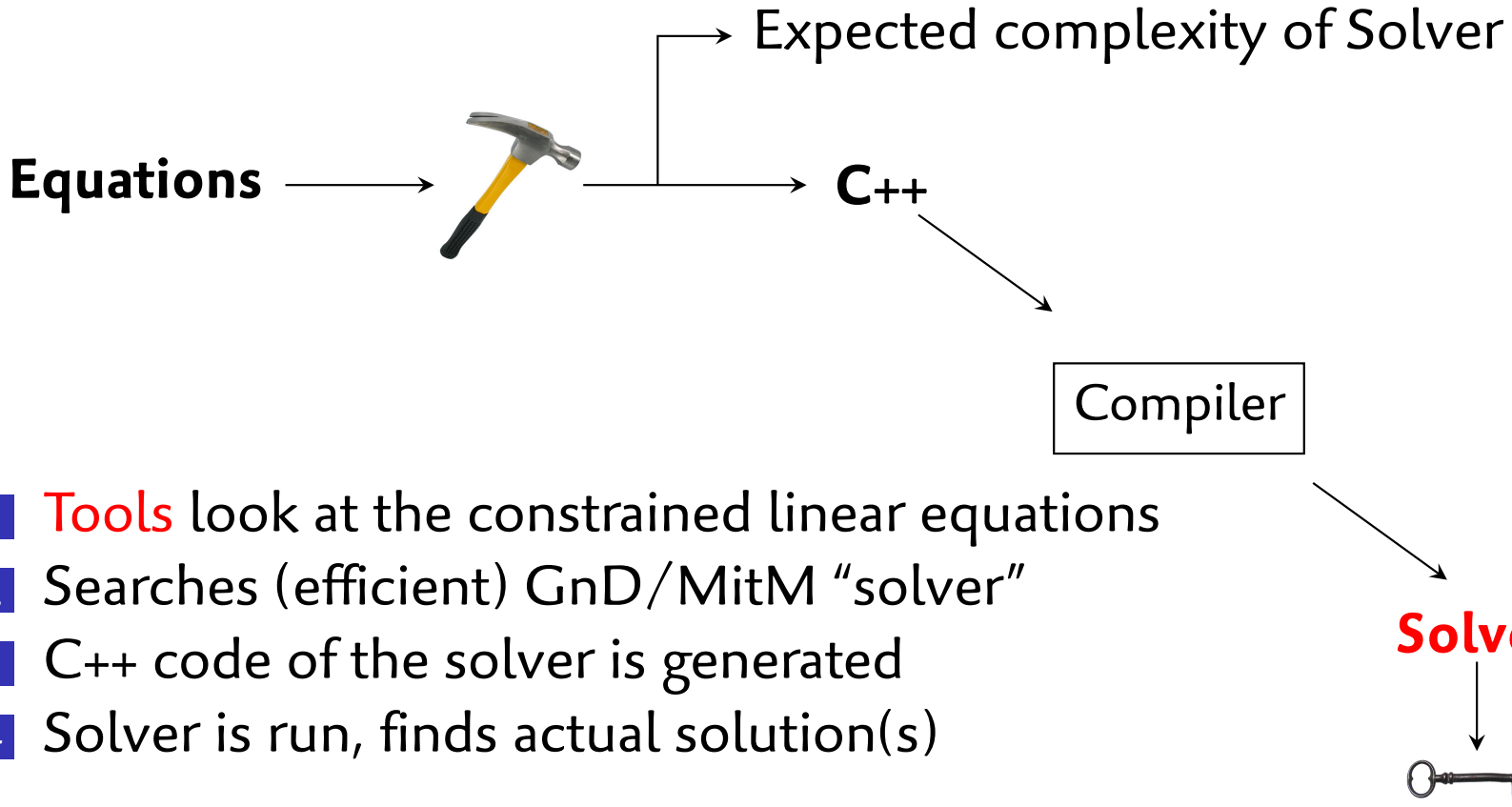
Simple in Theory...

- less guessed variables $\rightarrow$ it gets harder
- larger systems (more rounds) $\rightarrow$ it gets even harder

Simple in Theory...

- less guessed variables $\rightarrow$ it gets harder
- larger systems (more rounds) $\rightarrow$ it gets even harder

A Two-Phase Approach



1. Tools look at the constrained linear equations
2. Searches (efficient) GnD/MitM "solver"
3. C++ code of the solver is generated
4. Solver is run, finds actual solution(s)

In Cryptographic Terms...

- The Equations describe a cryptographic **Primitive**
- The Solver is an **Attack** against the primitive
- The Tools is a fully-automatic **Attack-Finder**

Finding Guess-and-Determine Attacks Automatically

The equations are **sparse**

All terms known except one $\rightarrow$ knowledge propagation

$$\text{e.g.} \quad x_i + S(z_j) + 03 \cdot z_k = 0$$

The equations are (essentially) **linear**

Gaussian elimination allows more knowledge propagation:

$$\text{e.g.} \quad \begin{cases} x_i + S(z_j) & +03 \cdot z_k & & +7\mathtt{f} \cdot u_\ell & = & 0 \\ 3\mathtt{d} \cdot x_j & +56 \cdot z_k & +S(v_r) & +9\mathtt{a} \cdot u_\ell & = & 0 \\ \mathtt{c2} \cdot y_s & +84 \cdot z_k & +\mathtt{cf} \cdot S(v_r) & & = & 0 \end{cases}$$

All terms known except one in a linear combination

A DPLL-Like Search Procedure

The DPLL procedure for SAT-solvers (big success story!)

- Propagate constraints as much as possible
- SAT $\rightarrow$ Return model
- When stuck, choose a variable and guess it
- UNSAT $\rightarrow$ Backtrack
- Tricks to prune the search-space (backjumping, learning)

The DPLL procedure for... Us?

- Propagate knowledge as much as possible
- Everything known $\rightarrow$ store solution and Backtrack
- When stuck, choose a variable and guess it
- Worse than best known solution $\rightarrow$ Backtrack
- Pruning tricks (subsumption tests, linear variables)

A DPLL-Like Search Procedure

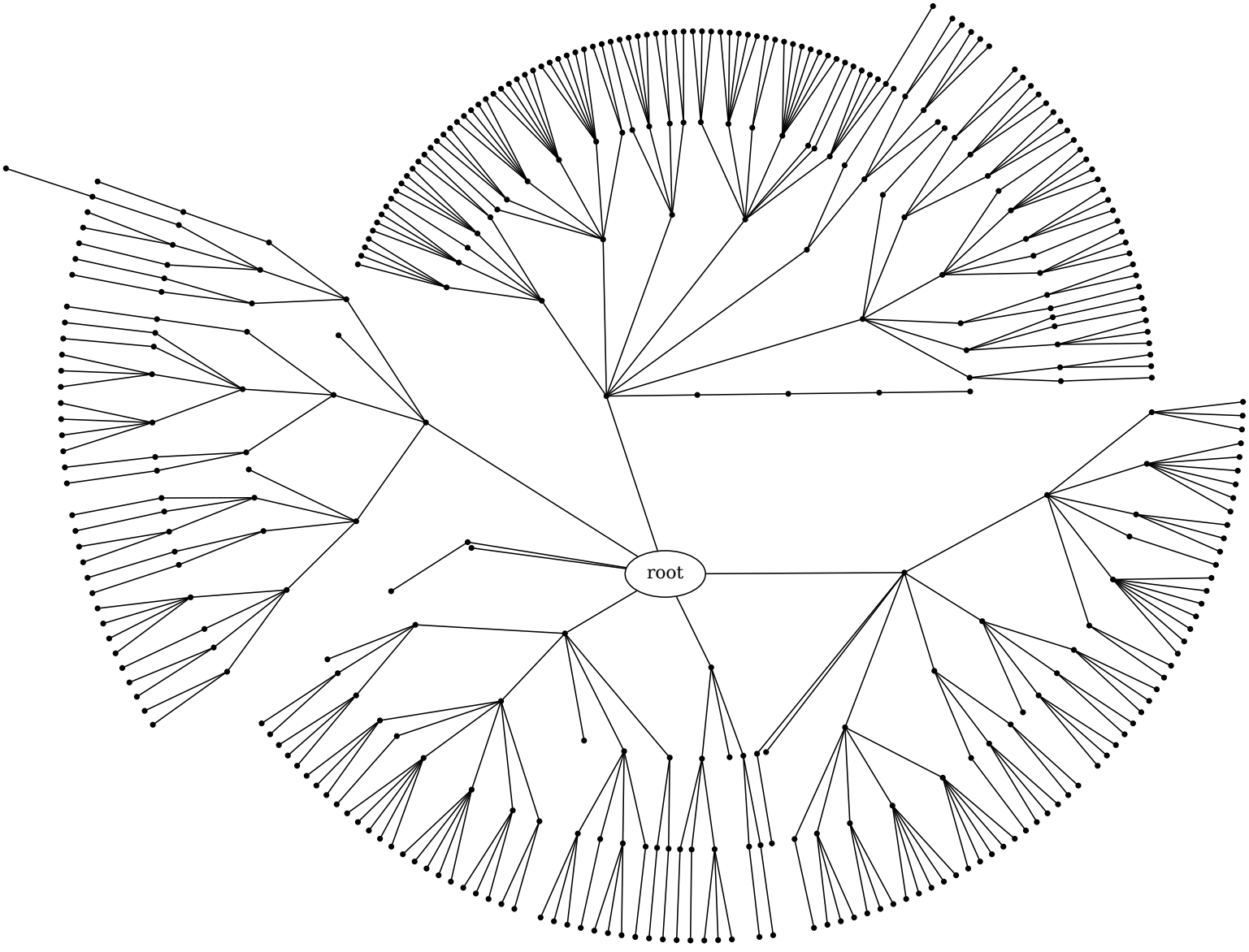
The DPLL procedure for SAT-solvers (big success story!)

- Propagate constraints as much as possible
- SAT $\rightarrow$ Return model
- When stuck, choose a variable and guess it
- UNSAT $\rightarrow$ Backtrack
- Tricks to prune the search-space (backjumping, learning)

The DPLL procedure for... Us?

- Propagate knowledge as much as possible
- Everything known $\rightarrow$ store solution and Backtrack
- When stuck, choose a variable and guess it
- Worse than best known solution $\rightarrow$ Backtrack
- Pruning tricks (subsumption tests, linear variables)

Attacking 1 round: Trace ($\geq$ 20 variables)



Implementation Details

Actual Implementation

- 5000 lines of OCaml
- Efforts put into efficiency
- Non-trivial sparse linear algebra
  - Triangular solver, re-echelonization
- Distributed: Home-made MapReduce on top of OcamlMPI
  - Run on $\approx$ 500 MIPS-like cores for a month without problem
- Generates C code of the solver

The Tool Put to Work

Results on round-reduced version of the AES-128

		Tool-found	**Human-found**
#Rounds	Data	Time	Time
1	1 pair	$\mathbf{2^{40}}$	2^{48}
2	1 pair	2^{80}	2^{96}
3	1 pair	2^{120}	not found

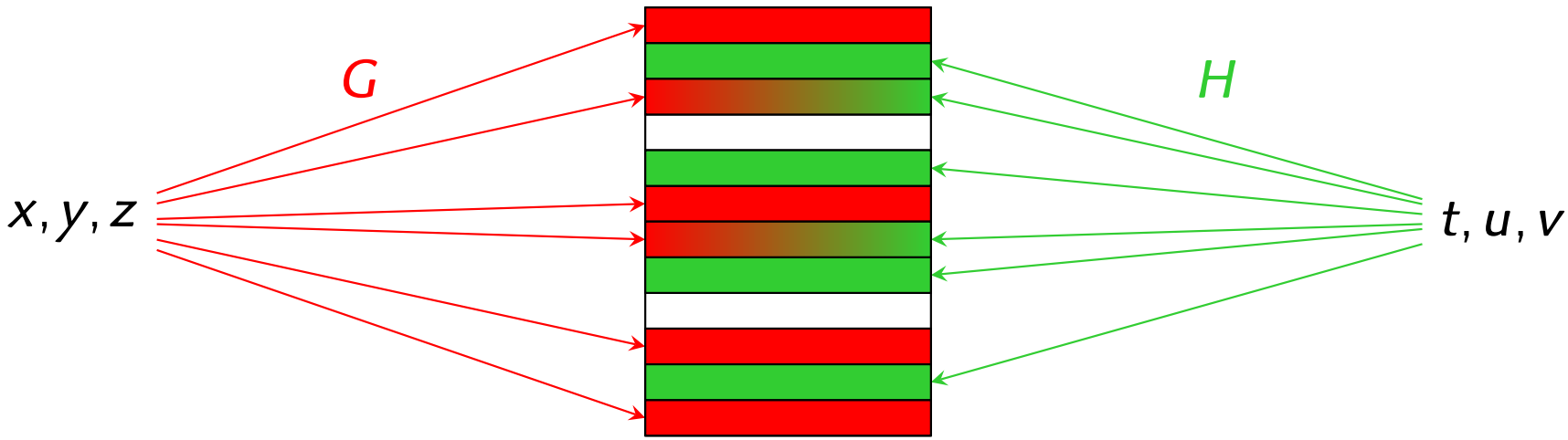
- The 1-round attack generated by the tool does work in 18h
- These attacks are guaranteed to be the best in this category

$\Longrightarrow$ ($\leq$ 3 round)-AES: Exhaustively explored the search space

Finding Meet-in-the-Middle Solvers

Idea: Partition the Set of Variables in Two

$$F(x, y, z, t, u, v) = 0 \iff G(x, y, z) = H(t, u, v)$$



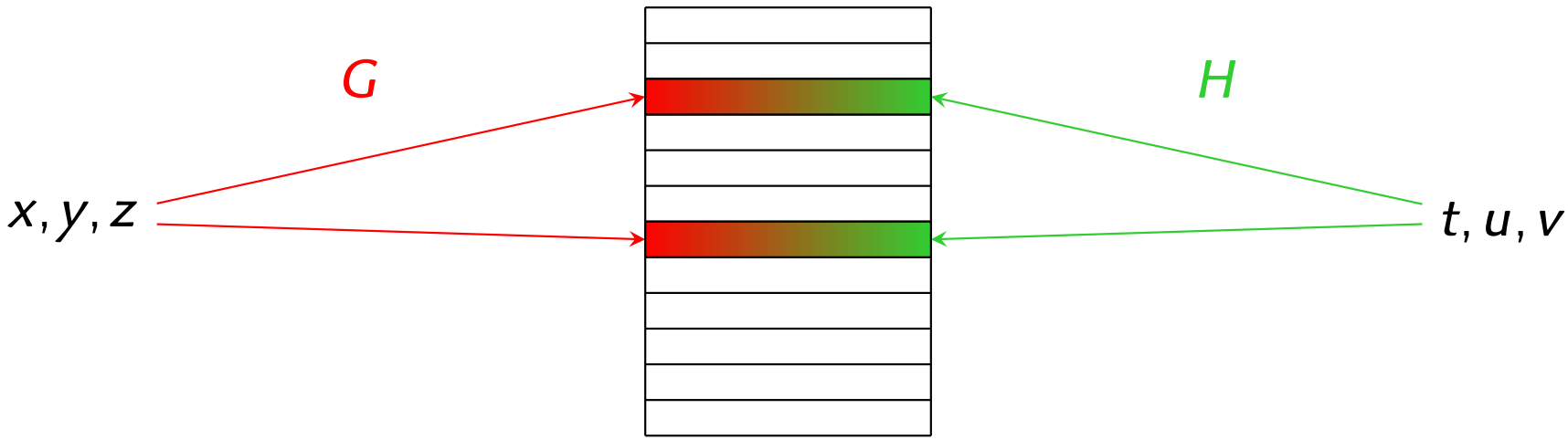
Meet-in-the-Middle Solver

- **for all** x, y, z, **store** $G(x, y, z) \mapsto (x, y, z)$ in a hash table
- **for all** u, v, t, **look-up** $H(u, v, t)$ in the hash table
  - On average one value of (x, y, z) per value of (u, v, t).

Finding Meet-in-the-Middle Solvers

Idea: Partition the Set of Variables in Two

$$F(x, y, z, t, u, v) = 0 \iff G(x, y, z) = H(t, u, v)$$



Meet-in-the-Middle Solver

- **for all** x, y, z, **store** $G(x, y, z) \mapsto (x, y, z)$ in a hash table
- **for all** u, v, t, **look-up** $H(u, v, t)$ in the hash table
  - On average one value of (x, y, z) per value of (u, v, t).

Variable Elimination

Suppose we have a partition $X \cup Y$ of the variables.

Question: are there equations in only X (resp. Y)?

- Intuition: **yes**, because equations are very sparse
- **How** to find them?
- Generic Problem known as **Elimination**
  - Want to get rid of some variable(s)
- Just selecting a subset of the initial system seems **bad**
- Polynomial systems: computation of Elimination Ideals
  - Gröbner Basis computation with ad hoc term order

A Compromise Solution

- **easy** to find the linear combinations with variables in X
  - Vector-space Intersection, **Polynomial** time complexity

Recursive Top-Down Approach

Objective: Find **Balanced** partition such that

$$F(x,y,z,t,u,v) = 0 \iff \begin{cases} G_1(x,y,z) & = & H_1(t,u,v) \\ G_2(x,y,z) & = & 0 \\ 0 & = & H_2(t,u,v) \end{cases}$$

Improved Solving Algorithm

- **for all** solutions of $G_2(x,y,z) = 0$
  - **Store** $G_1(x,y,z) \rightarrow (x,y,z)$ in a hash table
- **for all** solutions of $H_2(u,v,t) = 0$
  - **Look-up** $H_1(u,v,t)$ in the hash table
- Each match suggests a complete solution
- Problem reduces to solving two **sub-problems recursively**

Same problem
Smaller instan

Higher-Order Point of View

$$F(X,Y)=0 \iff \begin{cases} G_1(X) &= H_1(Y) \\ G_2(X) &= 0 \\ 0 &= H_2(Y) \end{cases}$$

In fact we have described a Combination operation:

- Algorithm $\mathcal{A}_1$ solves $G_2(X)=0$
- Algorithm $\mathcal{A}_2$ solves $H_2(Y)=0$

$\Longrightarrow$ "meet-in-the-middle" algorithm $\mathcal{A}_1 \bowtie \mathcal{A}_2$ for $F(X,Y)=0$

$$\mathcal{A}_1 : \texttt{solver}(X), \qquad \mathcal{A}_2 : \texttt{solver}(Y),$$
$$\bowtie : \texttt{solver}(X) \to \texttt{solver}(Y) \to \texttt{solver}(X \cup Y)$$

- "Properties" (complexity) of $\mathcal{A}_1 \bowtie \mathcal{A}_2$ easy to determine

$$T(\mathcal{A}_1 \bowtie \mathcal{A}_2) = T(\mathcal{A}_1) + T(\mathcal{A}_2) + \#Sols(F)$$

$$M(\mathcal{A}_1 \bowtie \mathcal{A}_2) = \max\left\{M(\mathcal{A}_1), M(\mathcal{A}_2), \min\left[\#Sols(G_2), \#Sols(H_2)\right]\right\}$$

Towards Solving the Whole Problem

The actual difficulty...

... is to find a nice partition $X \cup Y$.

- **Bottom-up** approach
- Start from trivial algorithms that enumerate a single variable
  - Usually no equation in a single variable

  $\Longrightarrow$ Solutions = $\mathbb{F}_{256}$

- Combine them until **saturation**, throw bad/redundant ones

Just like

- Buchberger's algorithm for Gröbner Bases
- Knuth-Bendix completion for equational theories
- Resolution/paramodulation/whatever for 1st-order logic

The Saturation Procedure

Simplified Version

1: **function** BESTSOLVERS($\mathbb{E}$)
2: $G \leftarrow \{\text{BaseSolver}(x) : x \in \mathbb{X}\}$
3: $\mathcal{P} \leftarrow \{(G_i, G_j) : 1 \leq i < j \leq |G|\}$
4: **while** $\mathcal{P} \neq \emptyset$ **do**
5: Pick $(\mathcal{A}_1, \mathcal{A}_2) \in \mathcal{P}$ and remove it from $\mathcal{P}$
6: $\mathcal{C} \leftarrow \mathcal{A}_1 \bowtie \mathcal{A}_2$
7: **if** $\mathcal{C}$ not subsumed by anything in G **then**
8: $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\mathcal{A}, \mathcal{C}) : \mathcal{A} \in G\}$
9: $G \leftarrow G \cup \{\mathcal{C}\}$
10: **end while**
11: **return** G
12: **end function**

- Patrick Derbez : Implementation, Improvements, Tricks, ...
- 10'000 lines of C

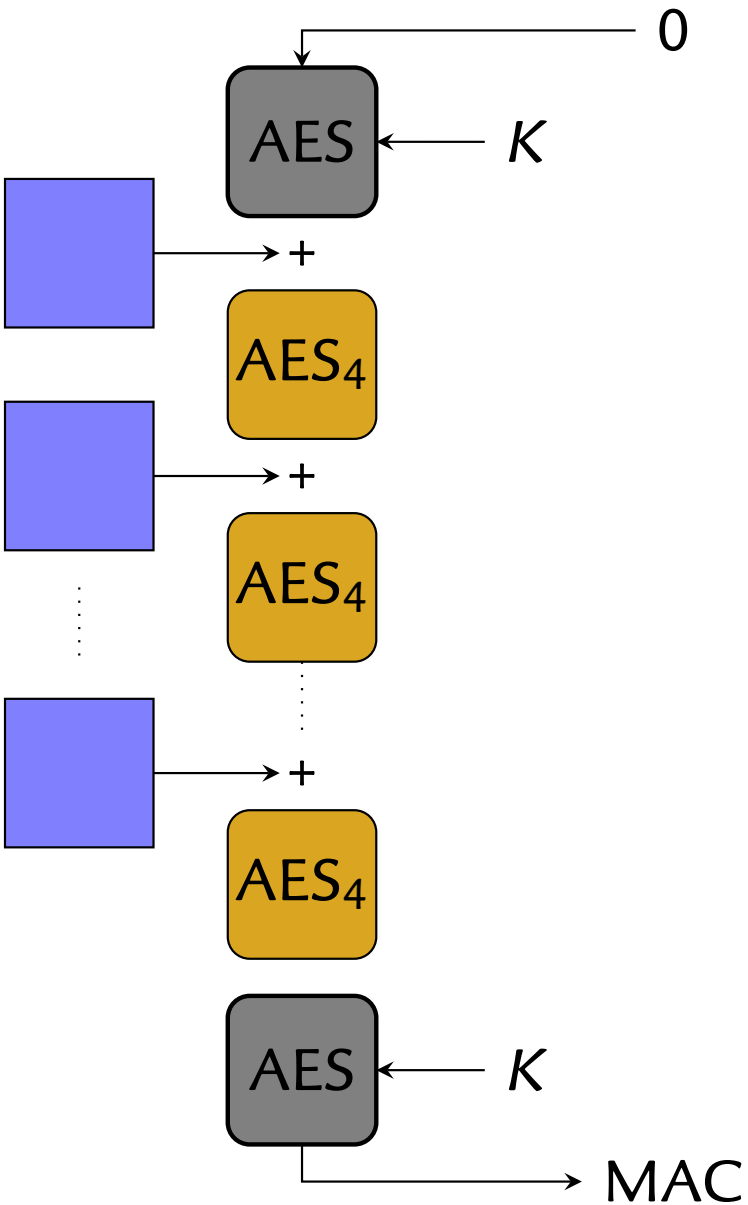
Results (Reduced AES)

Attacks on round reduced version of the AES-128

		Tool-found	Human-found
#Rounds	Data	Time	Time
1	**1 KP**	**2^{32}**	**2^{48}**
2	1 KP	2^{64}	2^{80}
2	**2 KP**	**2^{32}**	**2^{48}**
2	**2 CP**	**2^{8}**	**2^{28}**
3	1 KP	2^{96}	
3	**2 CP**	**2^{16}**	**2^{32}**
4	1 KP	2^{120}	
4	2 CP	2^{80}	2^{104}
4	**4 CP**	**2^{32}**	
4	5 CP		2^{64}
4.5	1 KP	2^{120}	

The attacks that are practical have been actually run and verified

Description of Pelican-MAC



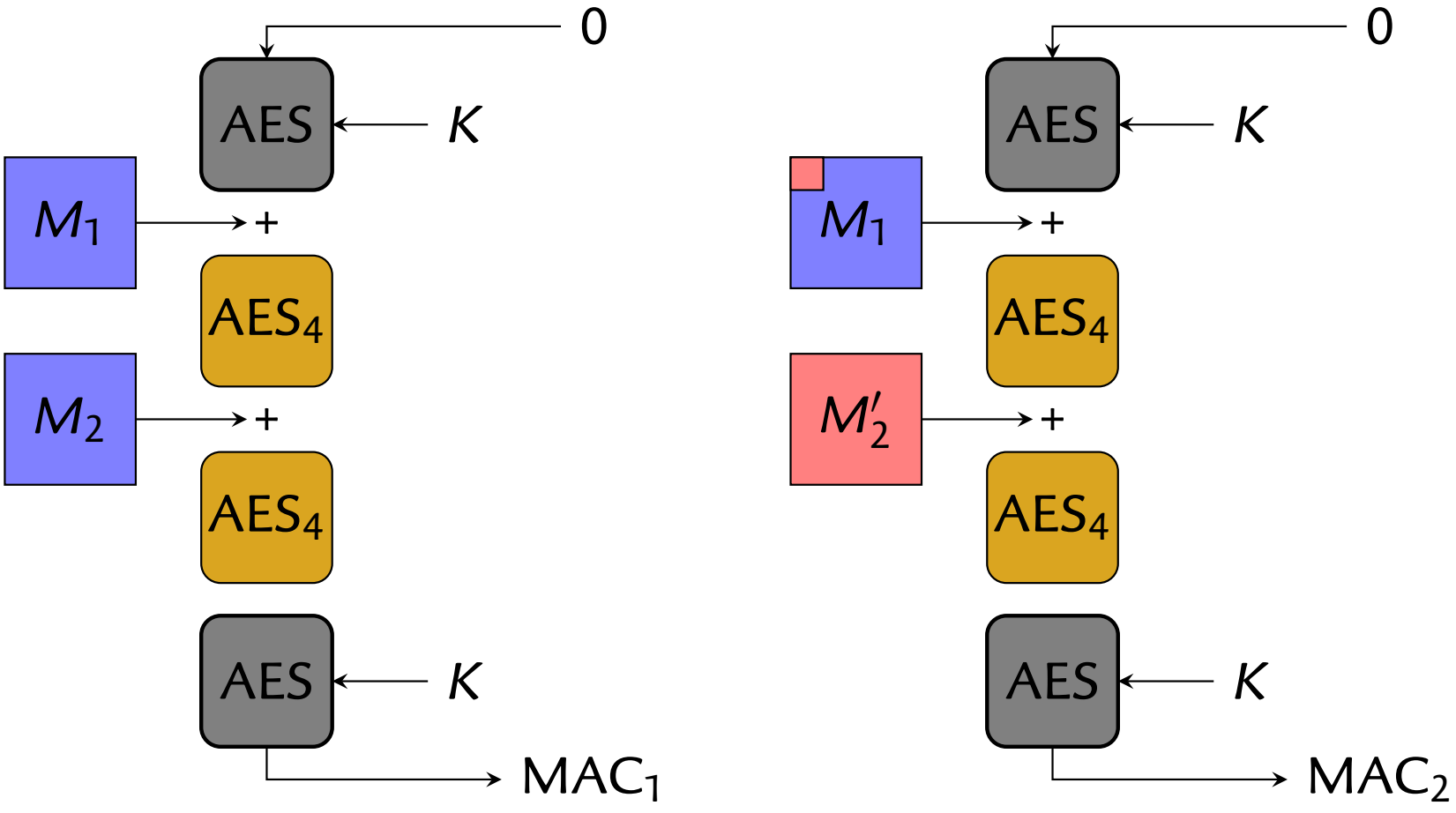
- MAC based on the AES
- Also by Rijmen & Daemen

- "Provably" secure up to 2^{64}
- Best known attack in $2^{85.5}$

- Initial state randomized with K
- 16-byte message block XORed
- 4 keyless AES rounds
- Finalization: full AES

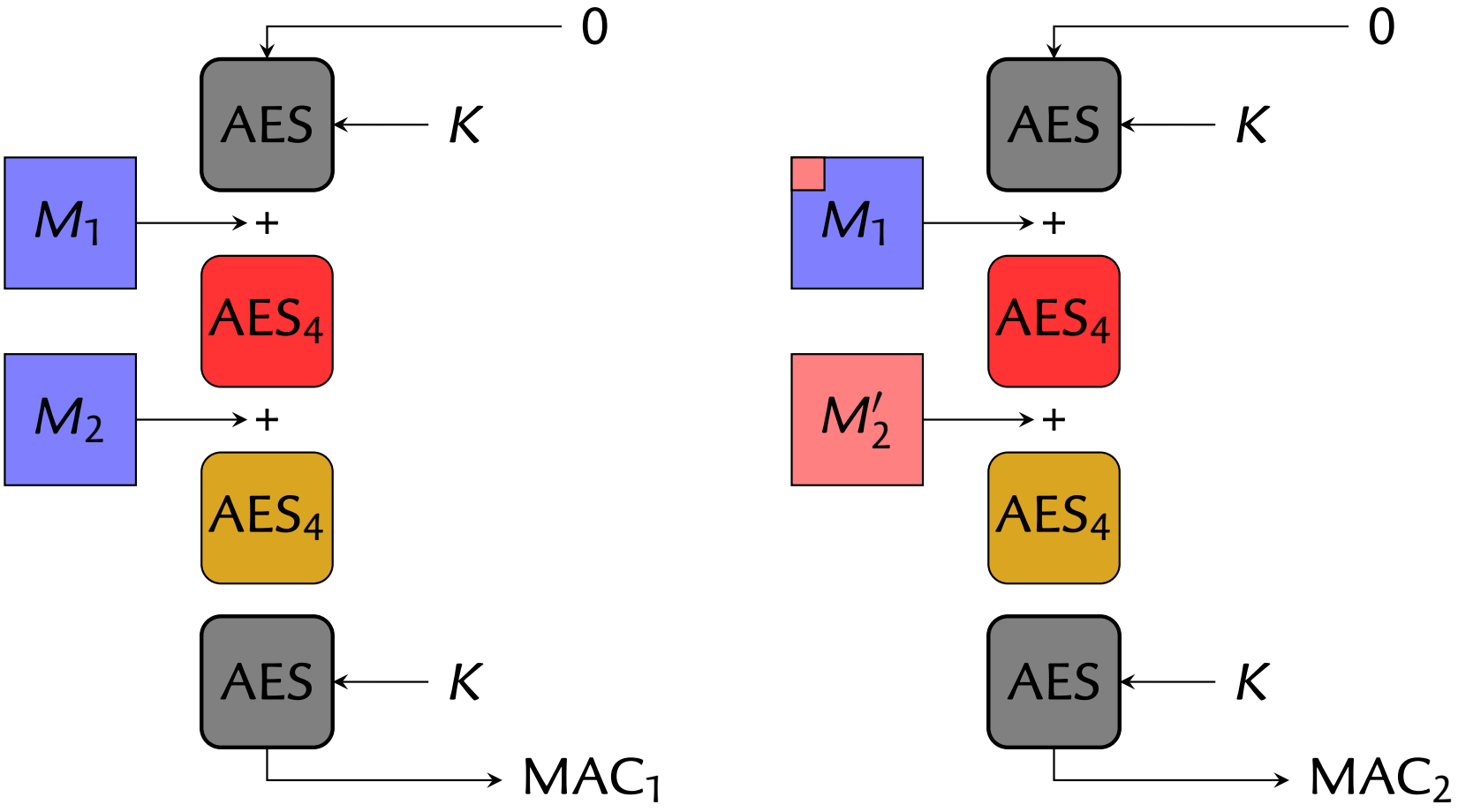
- Knowing the state $\rightarrow$ forgeries

Pelican-MAC: Finding an Internal State Collision



1. Pick random M_1
2. Try 2^{64} random M_2 and 2^{64} random M'_2
3. Look for $\mathrm{MAC}_1 = \mathrm{MAC}_2$

Pelican-MAC: Finding an Internal State Collision



1. Pick random M_1
2. Try 2^{64} random M_2 and 2^{64} random M'_2
3. Look for $\mathrm{MAC}_1 = \mathrm{MAC}_2$

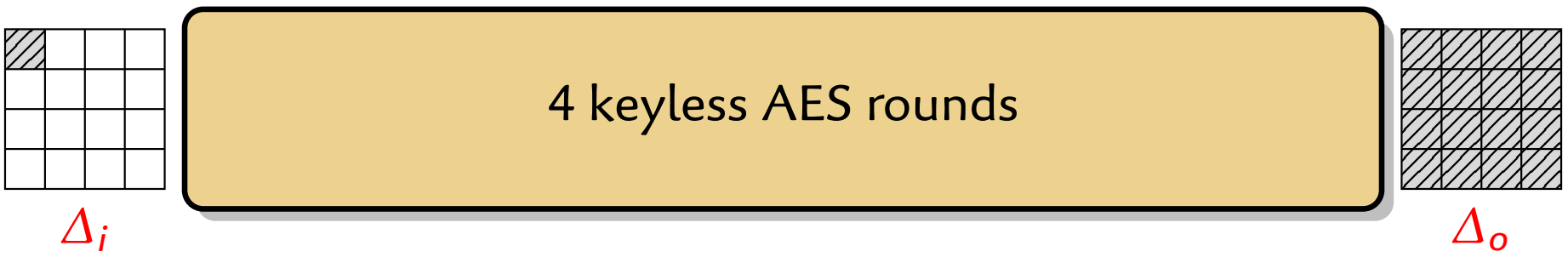
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

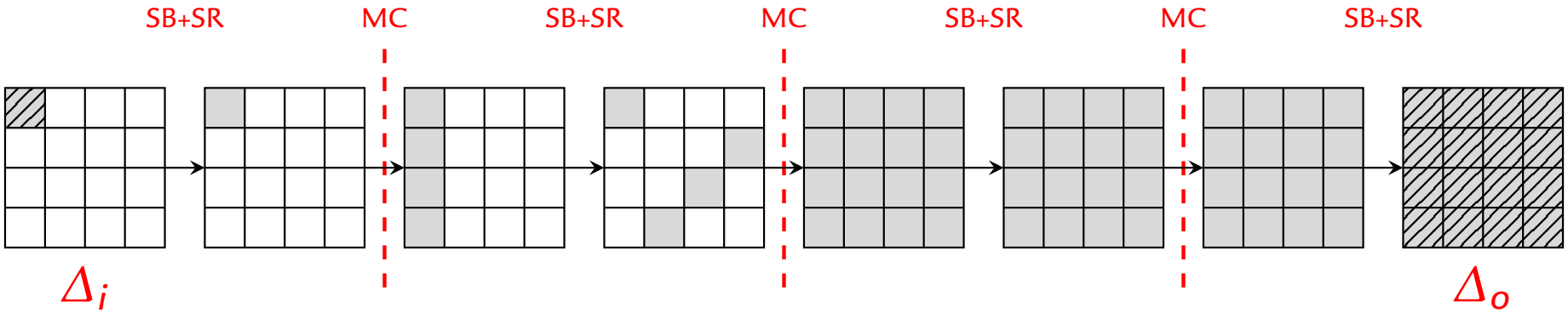
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

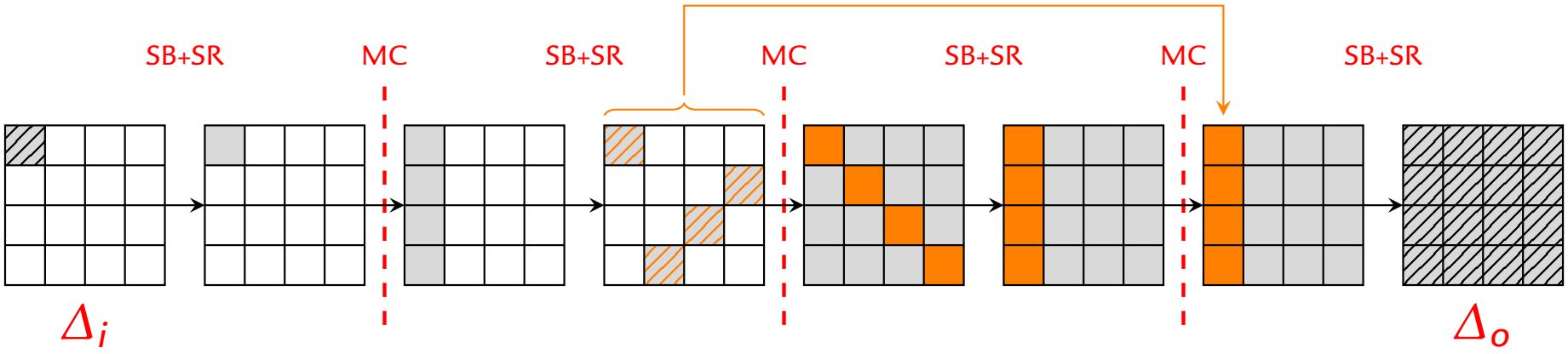
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

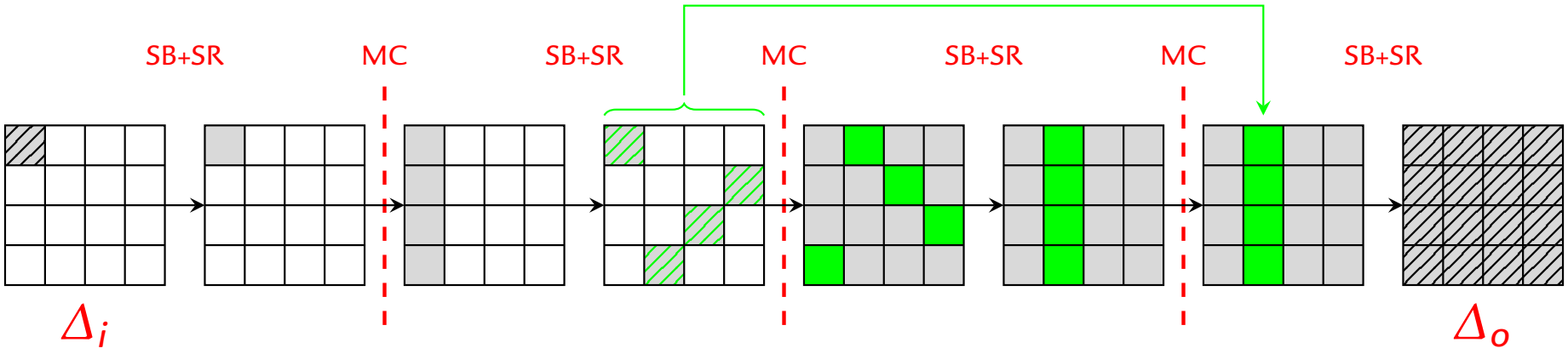
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

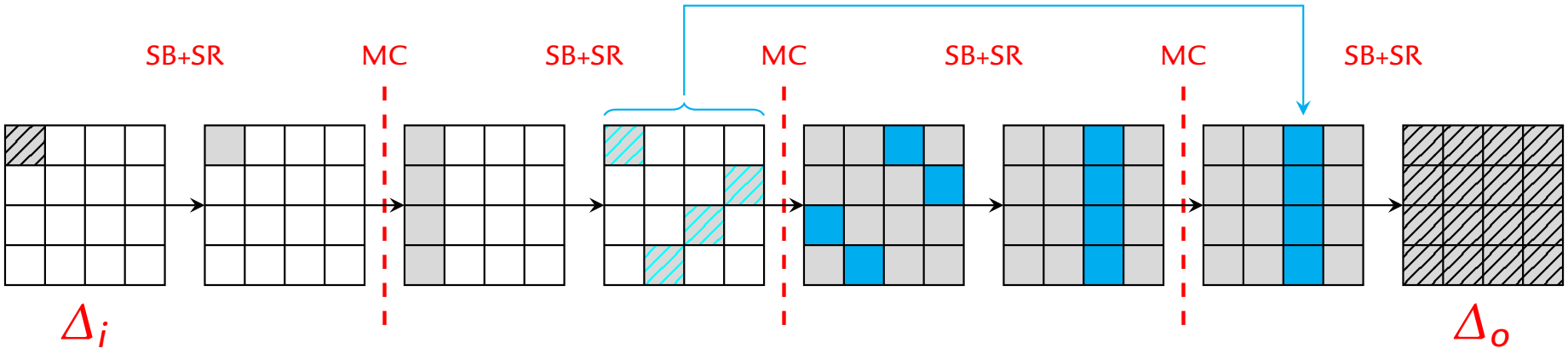
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

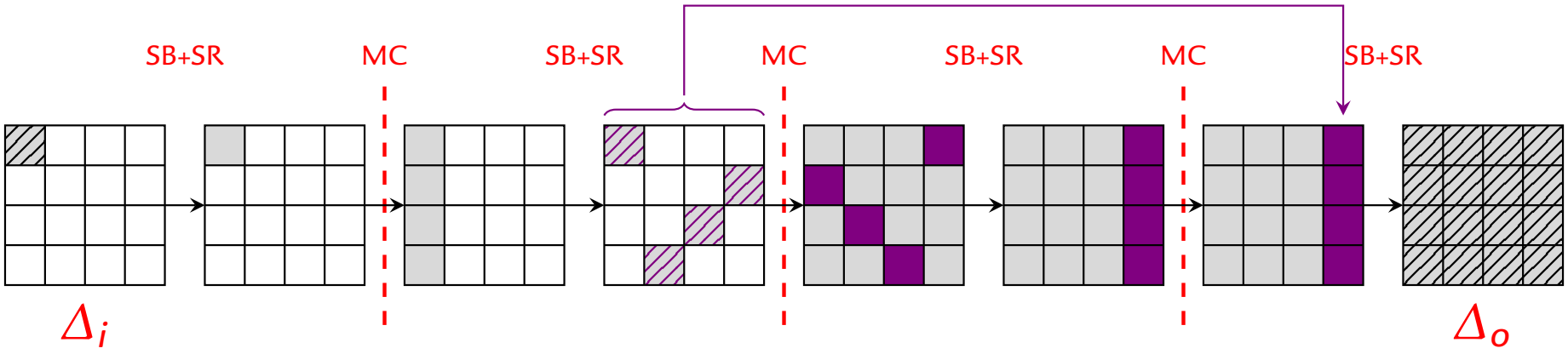
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

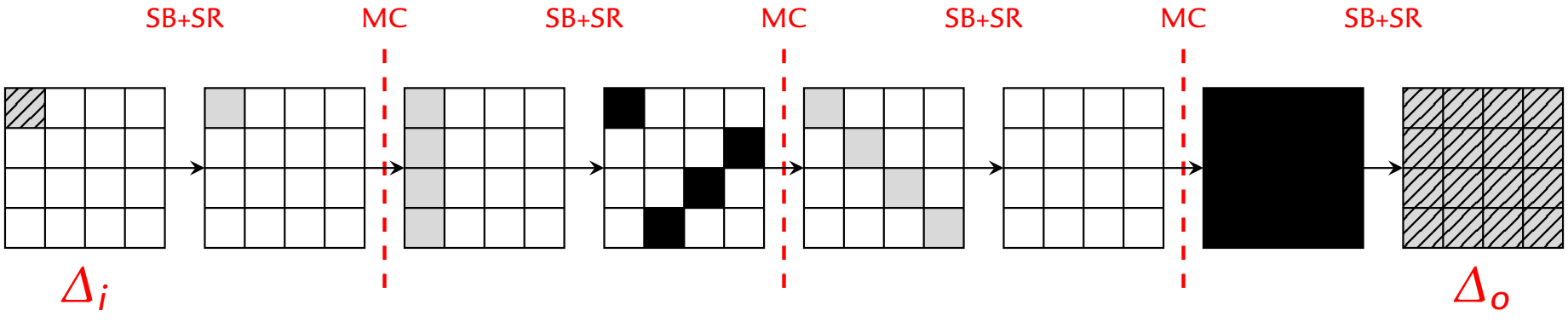
Results: Pelican-MAC

Attack Plan

- Query the MAC, find Internal Collision. Time = 2^{64}
- Solve

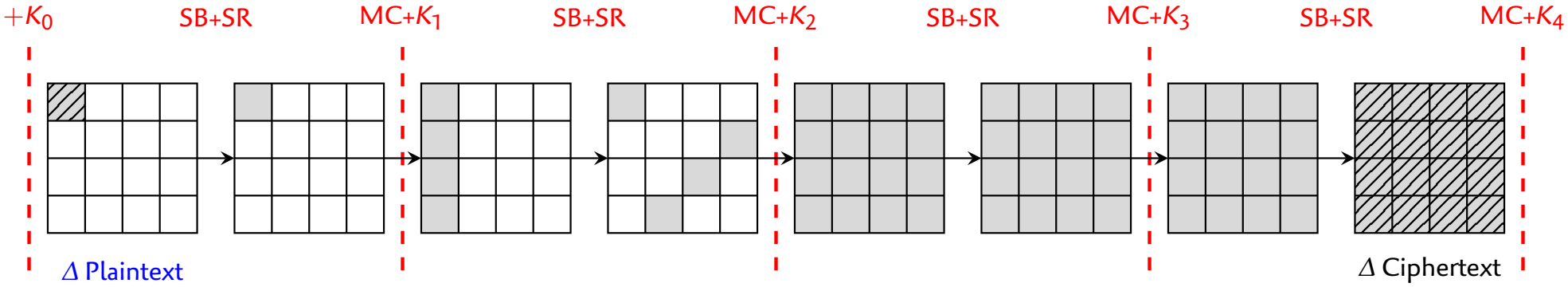
$$\mathrm{AES}_4(x \oplus \Delta_i) = \mathrm{AES}_4(x) \oplus \Delta_o$$

and recover x (the internal state). Time = 2^{32}



- 4 Hash Tables to build: 4×2^{32} opérations
- Isolate $\approx 2^{32}$ possible internal states

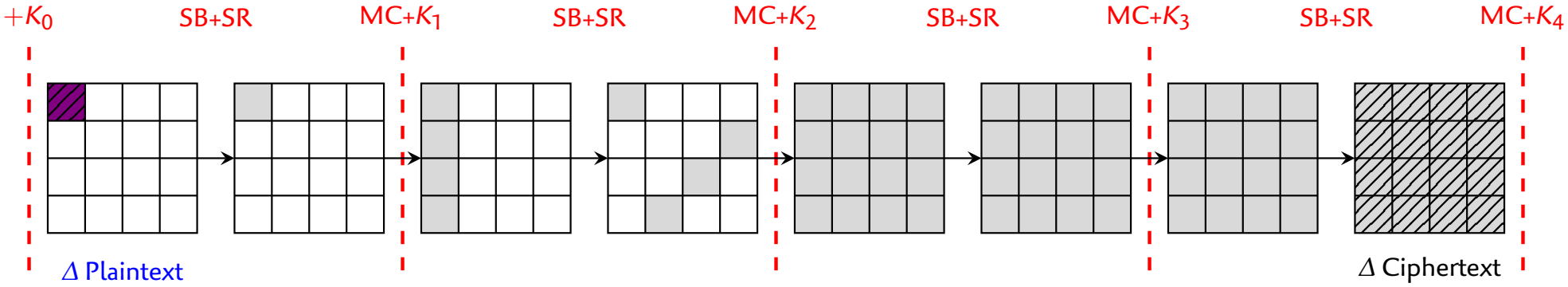
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● $\mapsto$ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

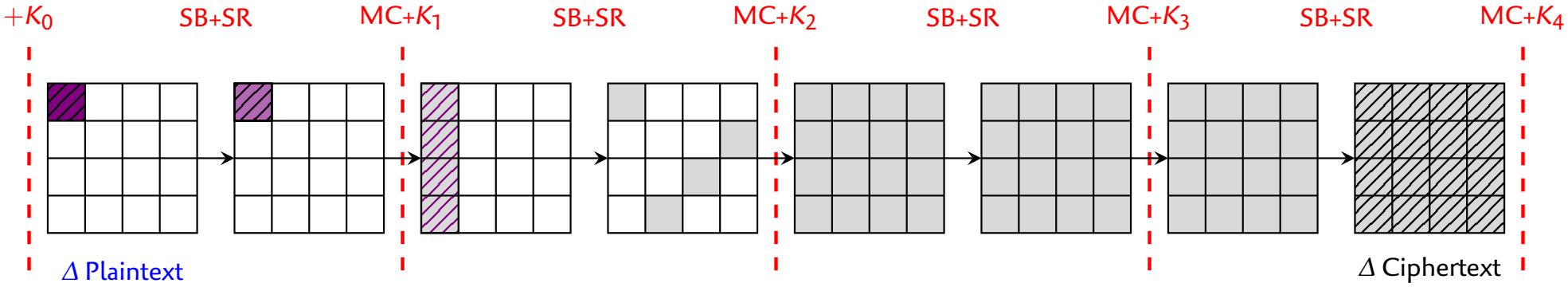
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● $\mapsto$ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

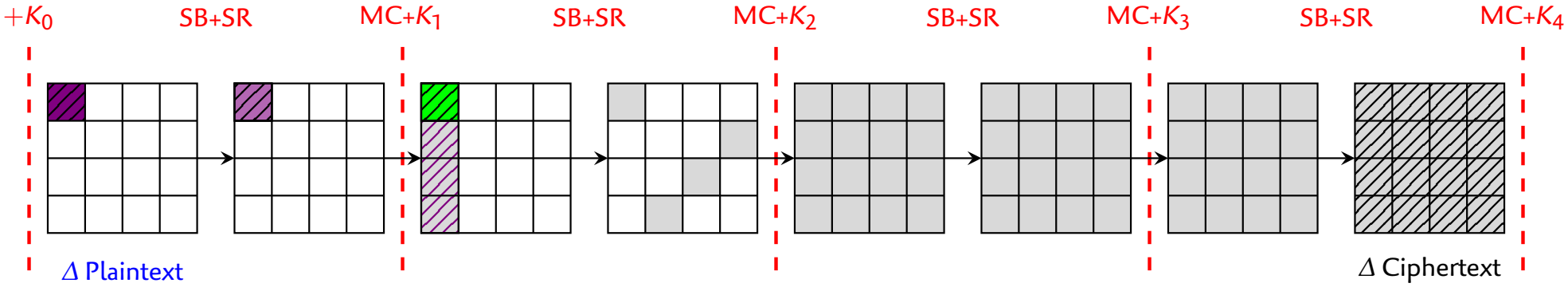
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

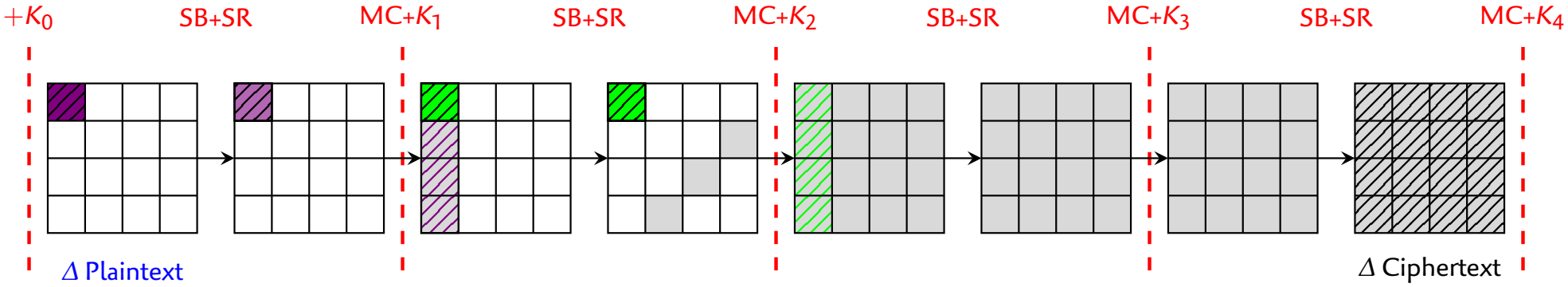
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● $\mapsto$ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

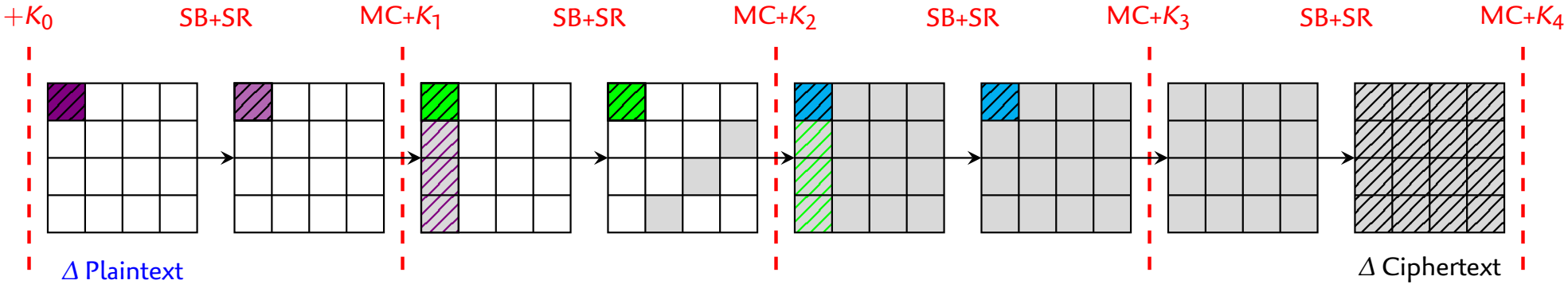
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

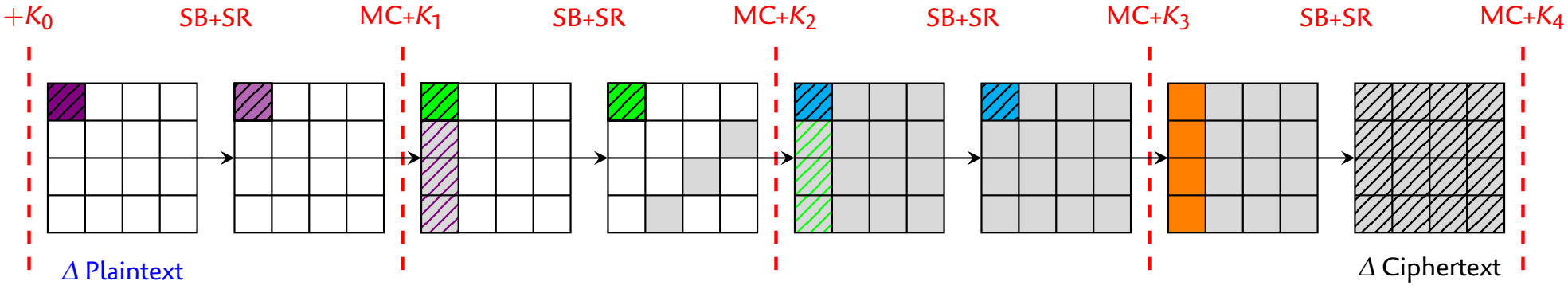
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

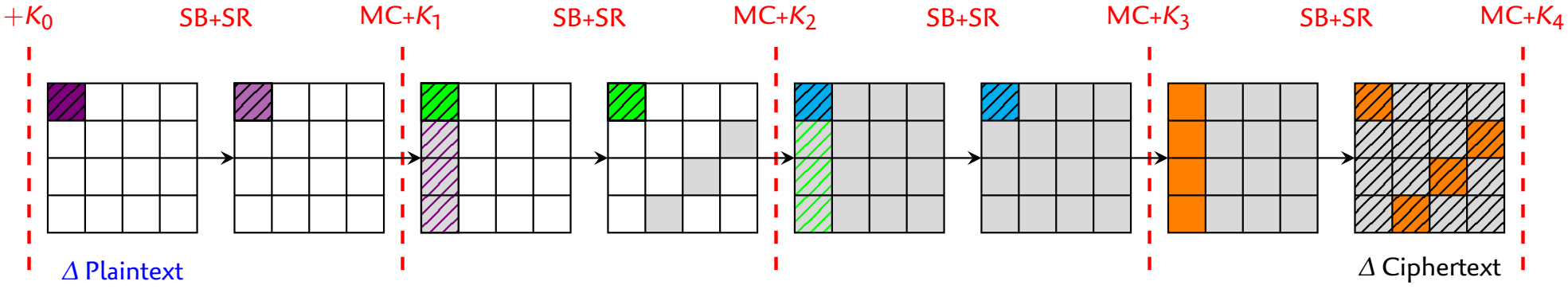
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

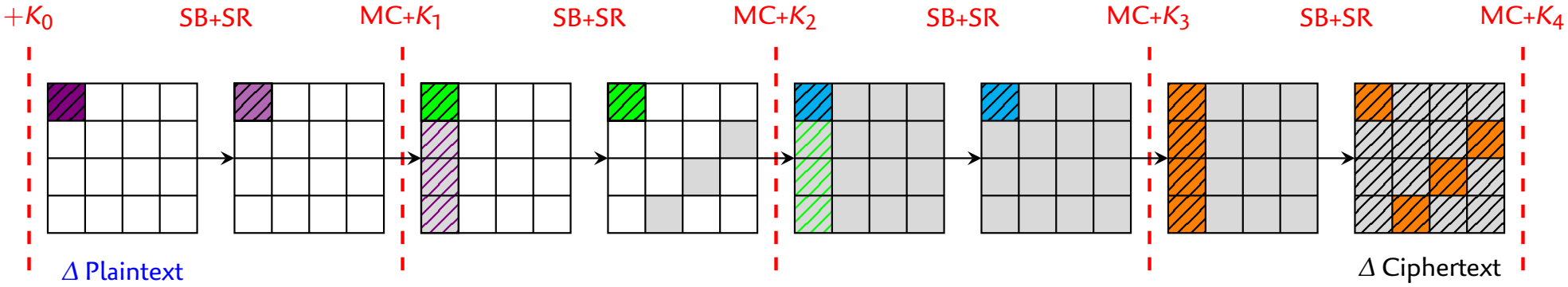
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

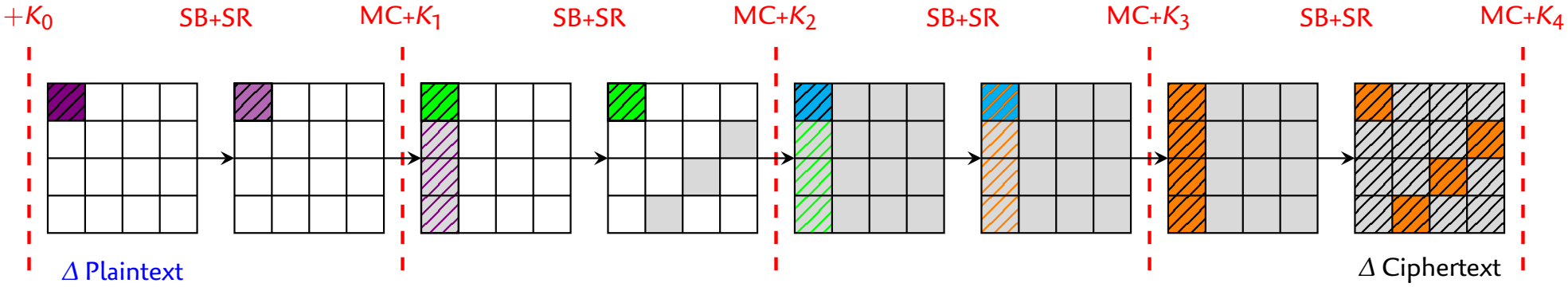
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● $\mapsto$ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

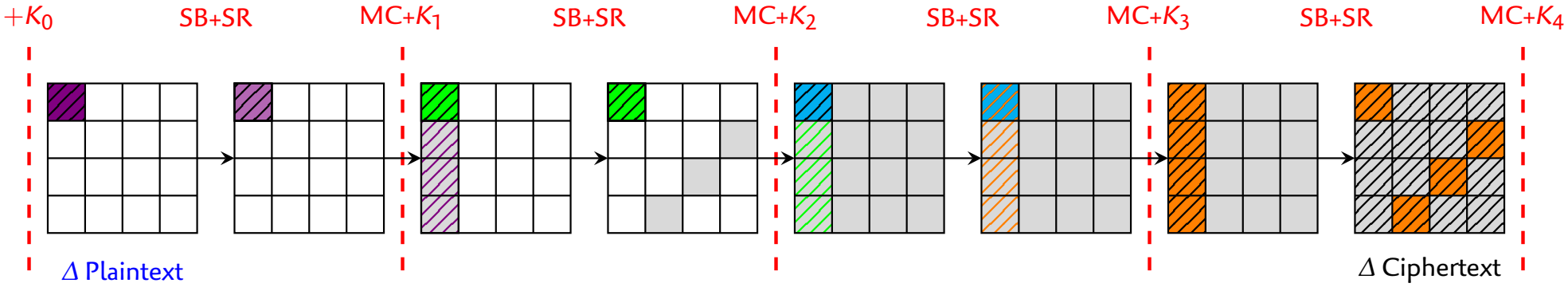
Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

Results: 4 AES Rounds, 4 Chosen Messages (1 active byte)



1. Build 16 hash tables : differences in ● ↦ (●, ●, ●)
2. Guess ●
3. Obtain differences before `MixColumn` on 4 bytes
4. Look-up 4 corresponding hash tables, check for matching ●
5. $\approx 2^8$ possible column ● pass the test
6. Try the $\approx 2^{32}$ possible combinations the 4 columns
7. deduce K_4, check

▸ Skip

Other Results

The method is somewhat generic, and applies to AES, SQUARE, PHOTON, SkipJack, LEX, Alpha-MAC, Pelican-MAC, etc.

LEX

- Stream Cipher by Biryukov, eStream Final Round candidate
  - Based on the AES, faster than CTR mode
  - Attacks by Dunkelman and Keller of complexity 2^{100}
- With the tool(s), new attacks found
  - Higher-order differential, complexity 2^{80}
  - Uses 80 tera-bytes of key-stream
  - success probability 2^{-5}

Summary

- Cryptanalysis-inspired methods to solve linear equations with an uninterpreted permutation
- Automatically find the best known (low data complexity) attacks on round-reduced AES, Pelican-MAC, LEX
- Can generate C++ code of the attacks
- Tools publicly available at:

`http://www.di.ens.fr/~bouillaguet/`

Future Work

- We are currently limited by the performance of our tools
  - exponential in every possible parameters
- We need to think on improving the search algorithms
  - have notions of critical/useless pairs
  - better subsumption tests
  - better pair selection heuristics
- We should exploit more thoroughly the input structure
  - similarity between the rounds
  - between concurrent encryption of several messages

And...

Thank You