Cryptanalyse des fonctions éponges

Séminaire de cryptographie - Université de Caen

Thomas Peyrin

Ingenico

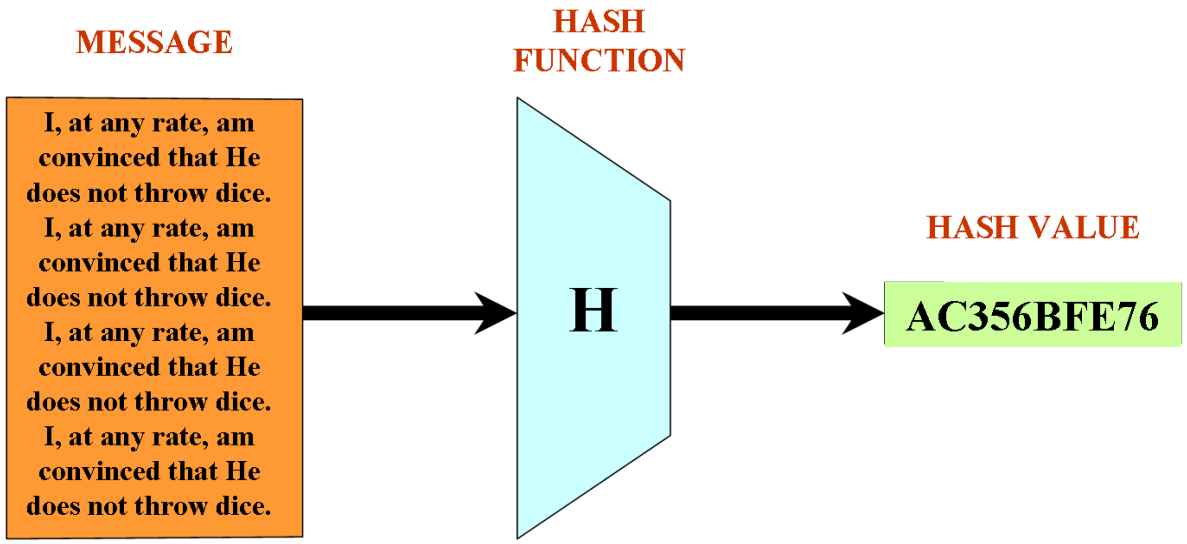
20 novembre 2008

Outline

Outline

Outline

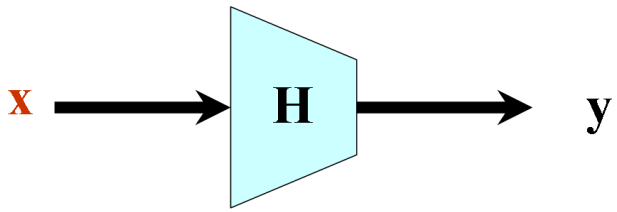
What is a hash function ?



- H maps an **input of arbitrary length** (the message M) to a **fixed length n-bit output** (typically $n = 128$, 160 or 256)
- no secret parameter
- generally regarded as part of symmetric key cryptography

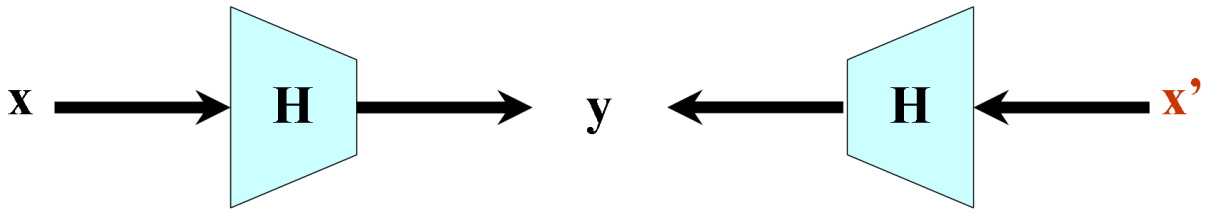
The security goals

- **pre-image resistance:** given an output challenge y, the attacker cannot find a message x such that $H(x) = y$ in less than $\theta(2^n)$ operations
- **2nd pre-image resistance:** given a challenge (x, y) such that $H(x) = y$, the attacker cannot find a message $x' \neq x$ such that $H(x') = y$ in less than $\theta(2^n)$ operations
- **collision resistance:** the attacker cannot find two messages (x, x') such that $H(x) = H(x')$ in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79])



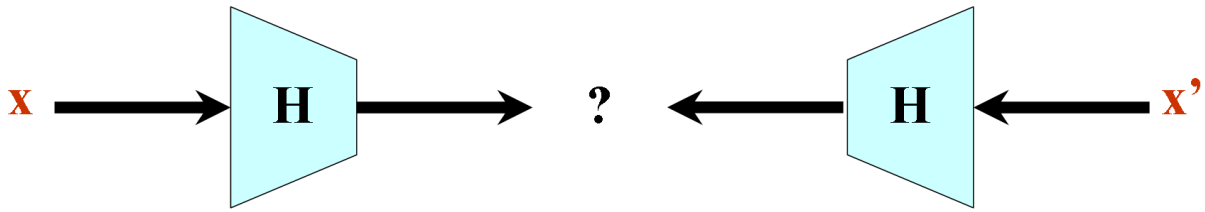
The security goals

- **pre-image resistance:** given an output challenge y, the attacker cannot find a message x such that $H(x) = y$ in less than $\theta(2^n)$ operations
- **2nd pre-image resistance:** given a challenge (x, y) such that $H(x) = y$, the attacker cannot find a message $x' \neq x$ such that $H(x') = y$ in less than $\theta(2^n)$ operations
- **collision resistance:** the attacker cannot find two messages (x, x') such that $H(x) = H(x')$ in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79])



The security goals

- **pre-image resistance:** given an output challenge y, the attacker cannot find a message x such that $H(x) = y$ in less than $\theta(2^n)$ operations
- **2nd pre-image resistance:** given a challenge (x, y) such that $H(x) = y$, the attacker cannot find a message $x' \neq x$ such that $H(x') = y$ in less than $\theta(2^n)$ operations
- **collision resistance:** the attacker cannot find two messages (x, x') such that $H(x) = H(x')$ in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79])



Applications

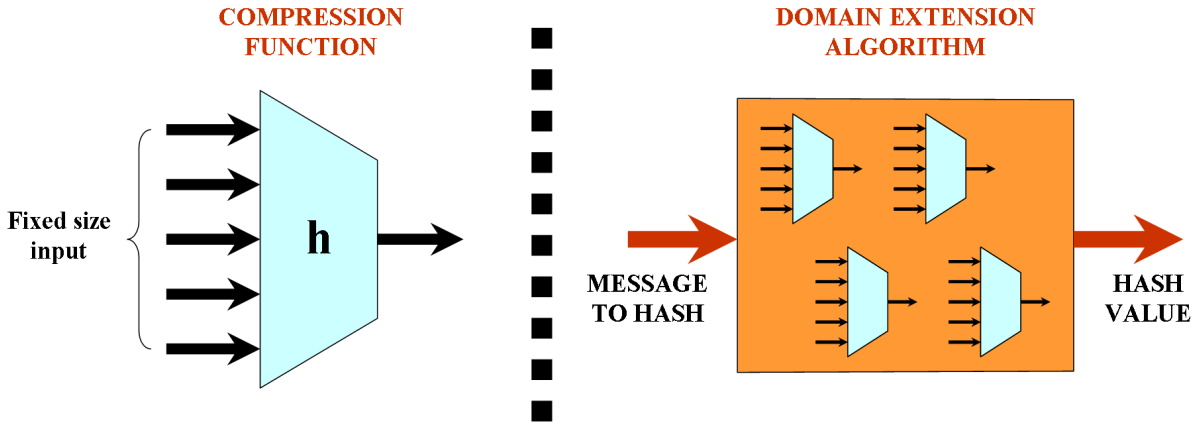
Hash functions are useful tools for many applications:

- **Digital signatures**: in the hash-and-sign paradigm, hash functions improve performance and security for digital signatures
- **Message Authentication Codes**: HMAC is built upon a hash function and is used in SSL/TLS, IPSec, ...
- **Password protection**: instead of storing all the passwords in a database, you can store the hash value of the passwords
- **Confirmation of knowledge/commitment**: if someone wants to prove that he knows some secret without revealing it, one can publish the hash value of this secret
- **Pseudo-random string generation/key derivation**: hash functions are known to destroy any structure that may exist in the input, while preserving to some extend the entropy

How to build a hash function ?

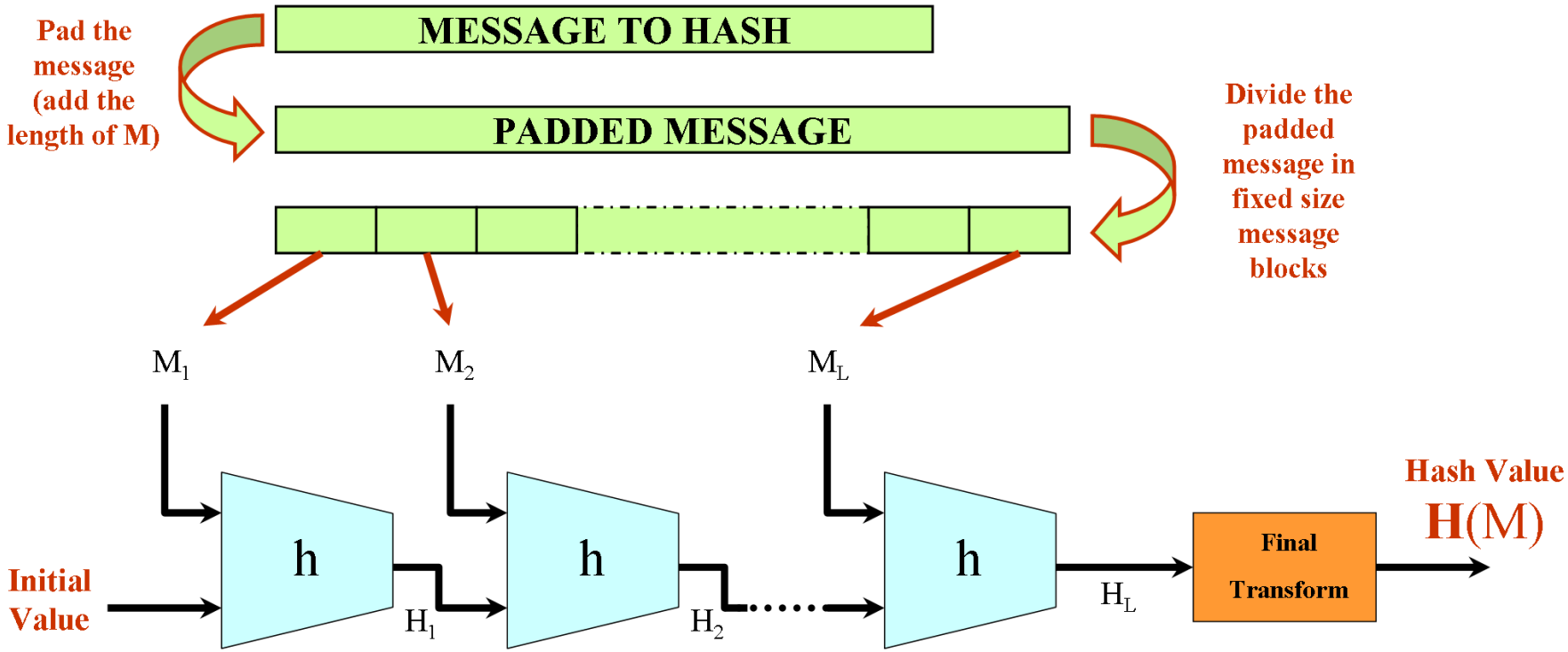
Almost all existing hash functions are built upon:

- **a compression function** *h*: a compressing function with fixed size input and output
- **a domain extension algorithm:** a (usually iterative) process using the compression function *h* in order for the hash function *H* to handle arbitrary length inputs



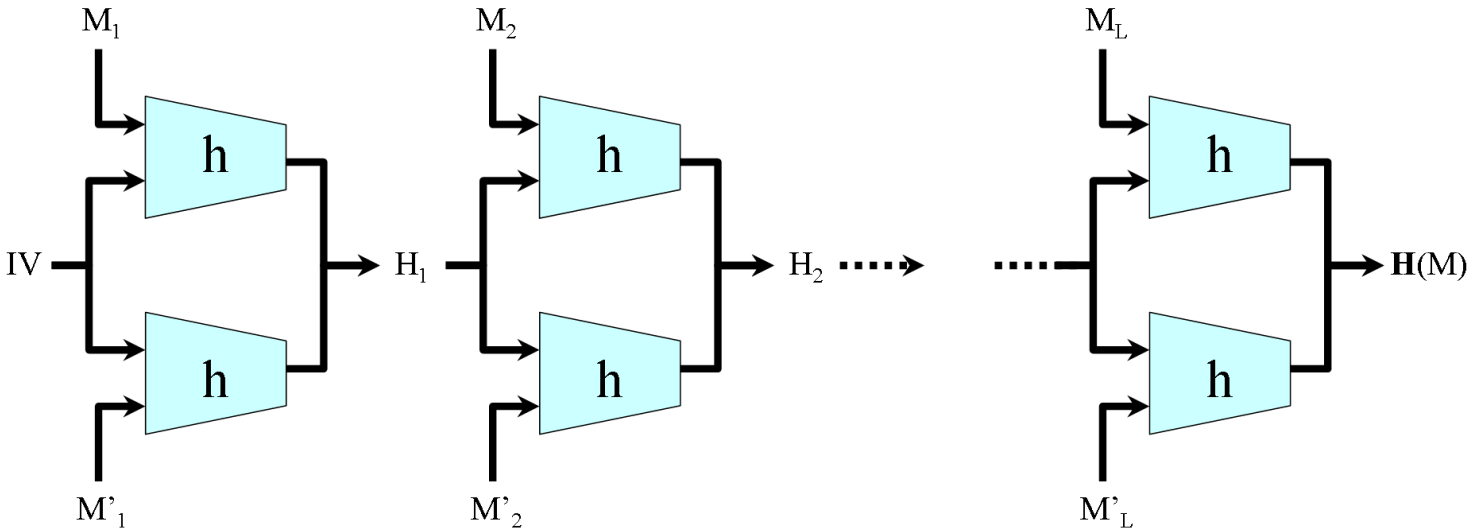
The Merkle-Damgård domain extension algorithm

The most popular domain extension algorithm is the **Merkle-Damgård iteration** [Merkle Damgård-89]



The Multicollision attack [Joux 04]

Multicollision attack applies on Merkle-Damgård : try to find k different messages so that they all map to the same output



- in the ideal case : $k!^{1/k} \times 2^{n(k-1)/k}$
- for Merkle-Damgård : $\log_2(k) \times 2^{n/2}$

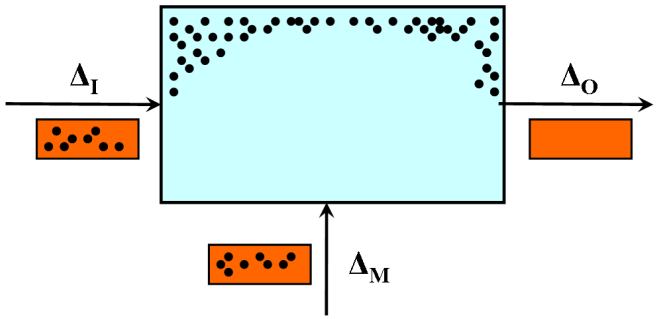
Groups of compression function designs

One can identify three methods to build a compression function:

- **from scratch:** very fast functions but one can trust their security only after much analysis by the crypto community (MD/SHA family: MD4, MD5, SHA-0, SHA-1, SHA-2, ...)
- **block cipher based:** proofs provided in the ideal cipher model, a little bit slower than from scratch
- **security related to a hard problem:** security proofs provided but tend to be slow compared to previous methods

Cryptanalysis of compression functions

- The cryptanalysis of compression functions uses a **differential path**, that specifies the exact difference masks in the message and in the internal state. It holds with a certain probability, which determines the core of the complexity of the attack
- The cryptanalysis of compression functions also uses the **freedom degrees** available: instead of trying random pairs verifying the input differential mask, one can adaptively choose input parts to improve the success probability



The security regarding collision resistance of various hash functions

Algorithm	Output size	Ideal Case	Attack Complexity
MD4 (1990)	128	2^{64}	2^{1}
MD5 (1992)	128	2^{64}	2^{30}
SHA-0 (1993)	160	2^{80}	2^{33}
SHA-1 (1995)	160	2^{80}	2^{60}
SHA-256 (2002)	256	2^{128}	no attack (yet!)
SHA-512 (2002)	512	2^{256}	no attack (yet!)

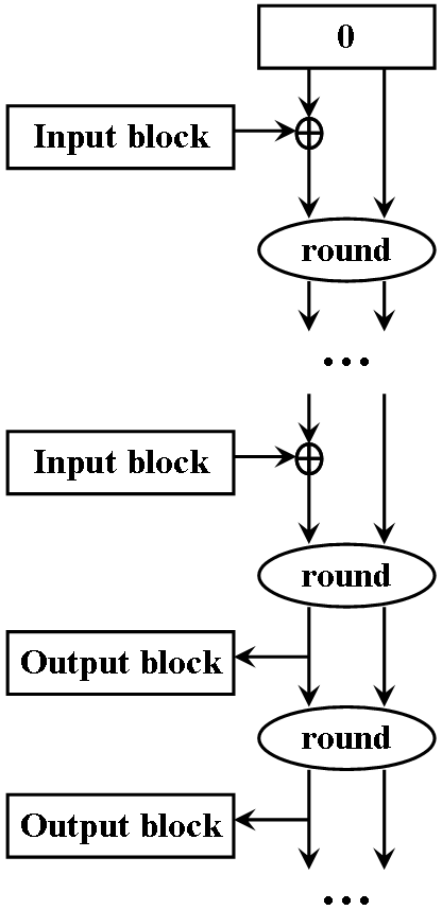
NIST's SHA-3 competition

- **who ?** just like the AES competition for block ciphers, the NIST is organizing a SHA-3 competition.
- **when ?** proposal submission deadline was set to October 31-th 2008, winner selection at the end of 2011.
- **why ?** SHA-1 is theoretically broken, soon a REAL collision will be found. SHA-256 and SHA-512 use the same design principles as MDx or SHA-x, so we need to be able to quickly jump to another algorithm. Moreover, SHA-256 and SHA-512 are vulnerable to generic attacks applying to Merkle-Damgård.
- **what ?** difficult question: we still don't know what we want !
  - random oracle lookalike or not ? MAYBE
  - one competition for each component ? NO
  - one hash function for each security property ? NO

We are much more ignorant of hash functions now that we were on block ciphers at the time of the AES competition :
Is it too early ?

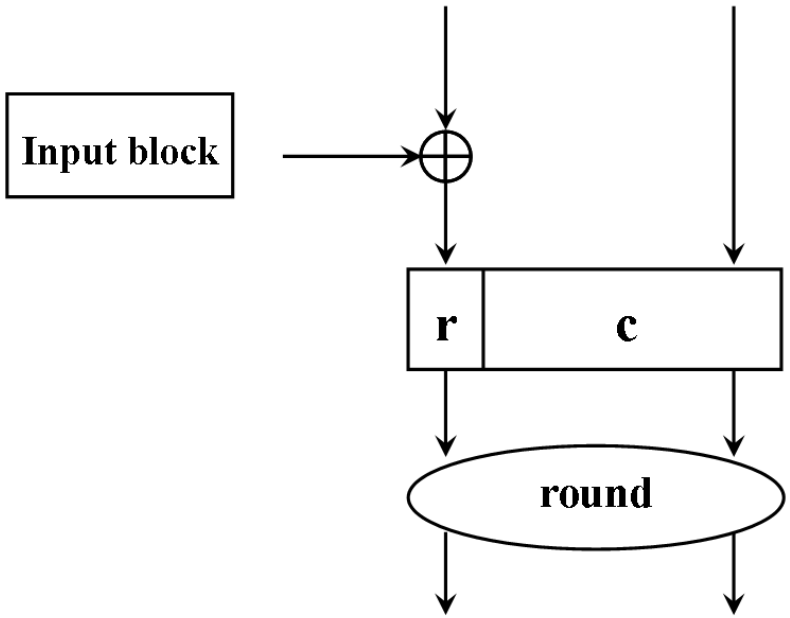
Outline

Sponge Functions



- sponge functions: a new iterative hash function (or stream cipher) framework.
- introduced by Bertoni, Daemen, Peeters and Van Assche in 2007.
- **idea:** absorb the message blocks (with padding) and squeeze the hash output blocks.
- use a unique fixed length round transformation (or a permutation).
- handy: variable output length.

Sponge Functions



- **c** represents the capacity.
- **r** represents the bit-rate.

Security Bounds (Eurocrypt 2008)

- **white box model**: the attacker has access to the internal round function
- use the indifferentiability framework from Maurer *et al.* (2007)
- **Theorem:** a random sponge can be differentiated from a random oracle only with probability $\simeq N(N+1)/2^{c+1}$, with $N < 2^c$, where N is the total number of calls to the internal round function
- generic attacks require $2^{c/2}$
- better results if the internal round function is a random permutation
- gives lower bounds for the attacks

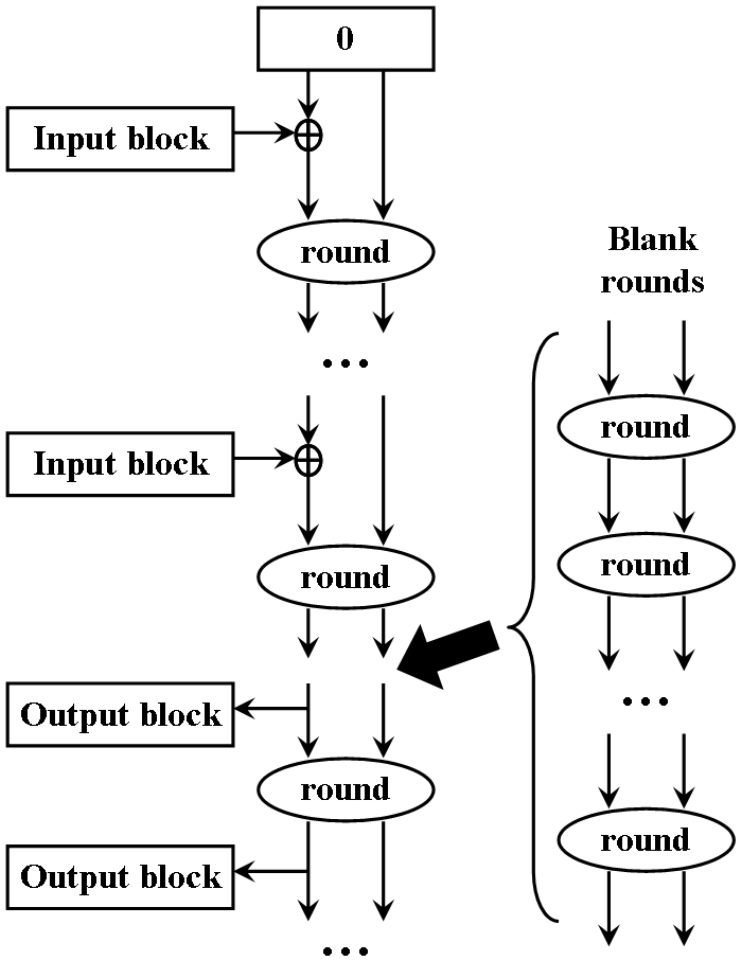
Why Sponge Functions ?

Sponge functions are interesting because:

- they allow new directions to build hash functions (no more MD-SHA-like or Merkle-Damgård constructions)
- they are built upon a permutation and not a compression function
- they give an easy way to thwart the generic attacks on iterated hash functions such as multicollisions, long 2nd-preimages, ...

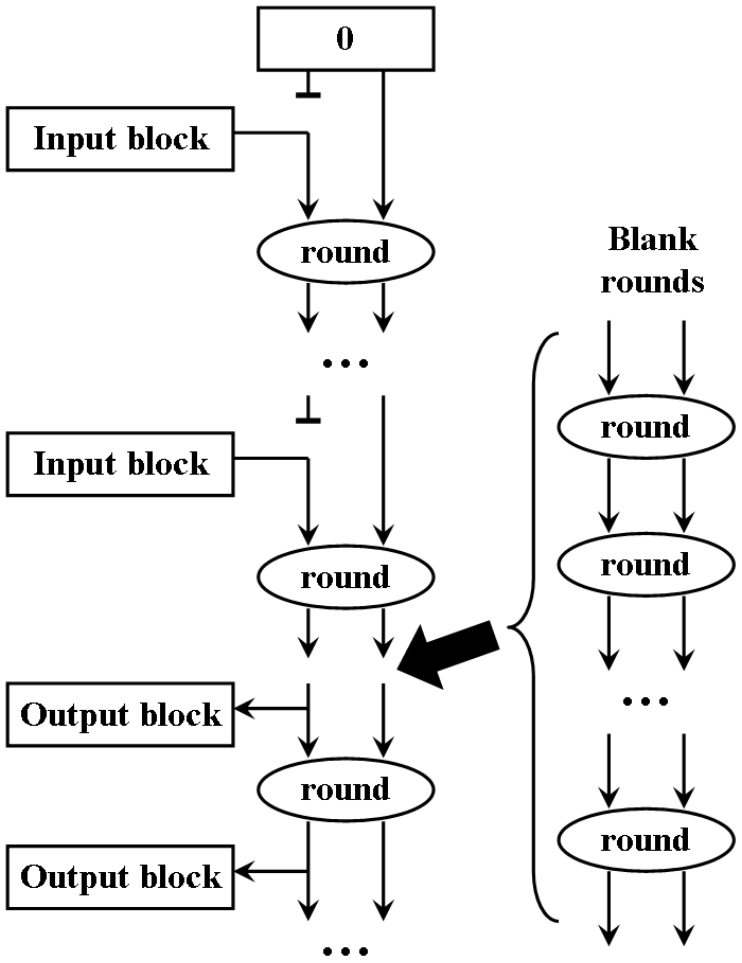
The idea of using a big internal state to avoid generic attacks on iterated hash functions was already pointed out by Joux and Lucks

Practical Sponge Functions



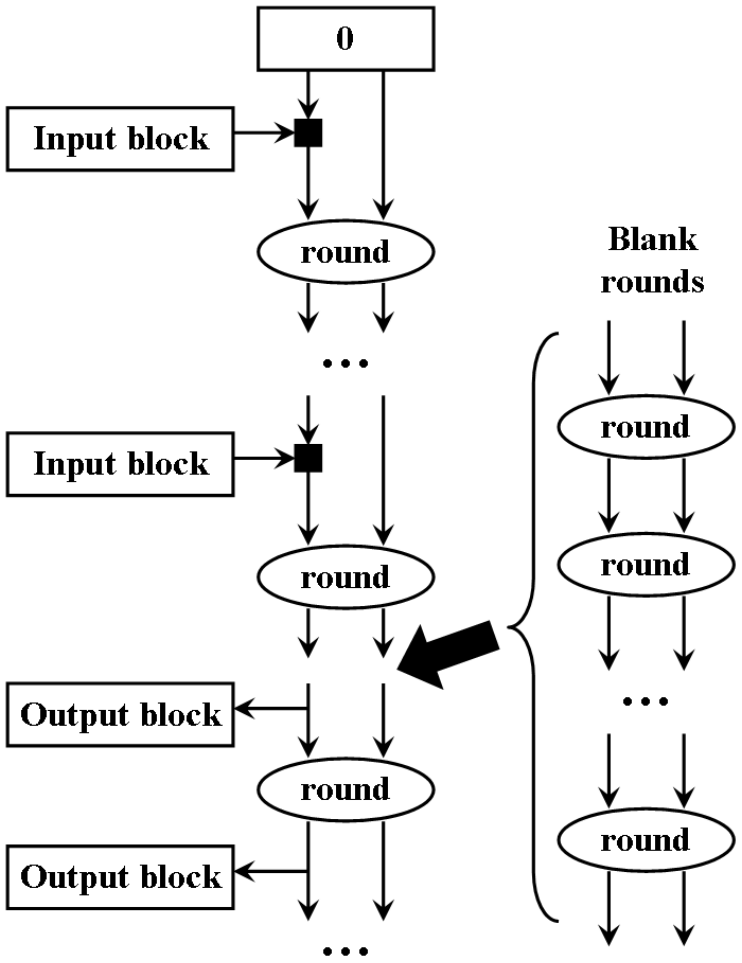
- in theory, the round function is ideal ... but not in practice
- seems relatively ok for collision resistance but seems weak for preimage resistance
- in practice, **we add blank rounds** (rounds without incoming message blocks)
- theory probably still applicable

Extended Sponge Functions



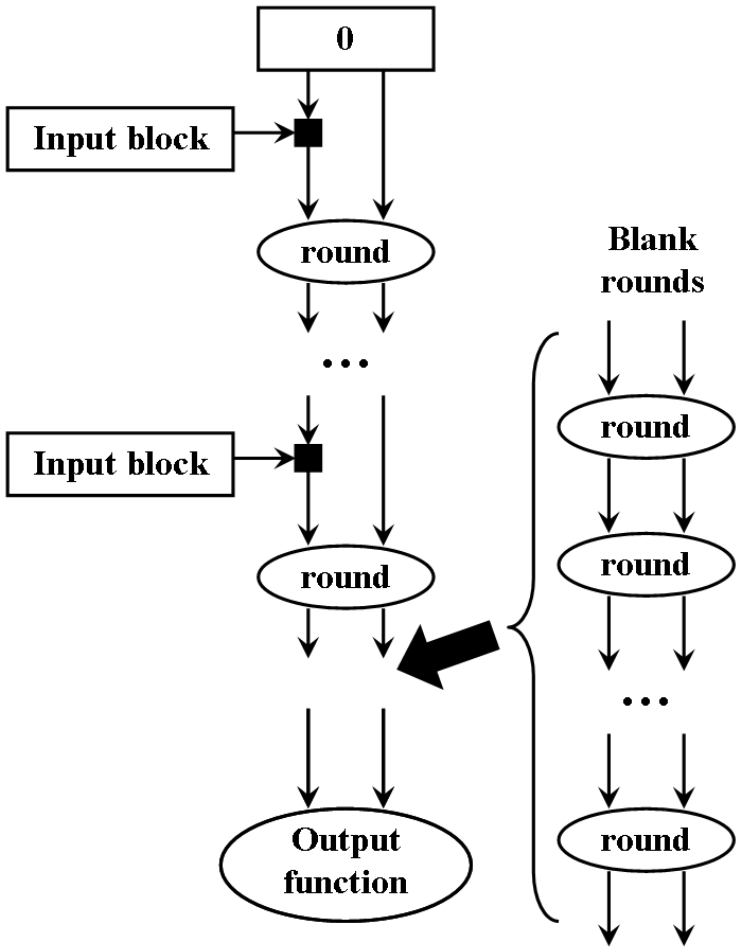
- we extend the original sponge functions framework
- instead of XORing the message blocks to the internal state, **we allow them to overwrite the corresponding areas**
- theory is no more applicable !
- an example : GRINDAHL

Extended Sponge Functions



- we extend the original sponge functions framework
- instead of XORing the message blocks to the internal state, **we allow them to overwrite the corresponding areas**
- theory is no more applicable !
- an example : GRINDAHL

Extended Sponge Functions



- we further extend the original sponge functions framework
- instead of squeezing the sponge to get output blocks, **we can directly truncate it**
- theory is no more applicable !
- an example : GRINDAHL

Outline

Outline

Slide Attacks for Block Ciphers

- slide attacks were introduced for block ciphers by Biryukov and Wagner in 1999
- **efficient against block cipher with a weak and periodic key schedule** (self-similarity of the cipher)
- independent of the number of rounds
- allows to mount **distinguishing attacks** or even **key recovery attacks**
- many improvements were later introduced

Slide Attacks for Block Ciphers

A n-bit block cipher E with r rounds is split into b identical rounds of the same keyed permutation F^i for $i = \{1, \ldots, b\}$:

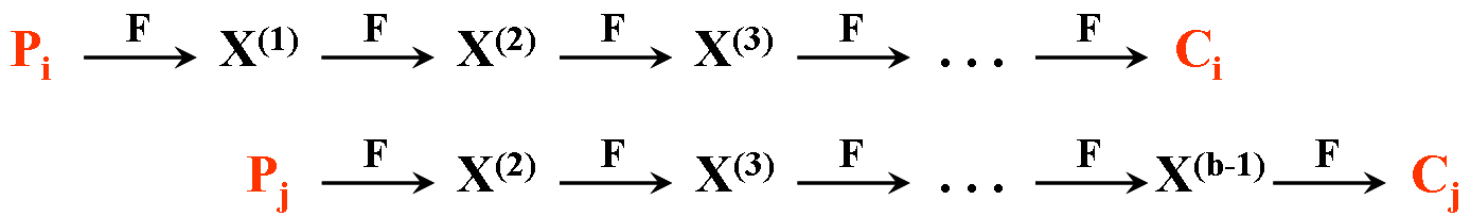
$$\begin{aligned} E &= F^1 \circ F^2 \circ \cdots \circ F^b \\ &= F \circ F \circ \cdots \circ F \end{aligned}$$

A plaintext P_j is then encrypted as:

$$P_j \xrightarrow{F} X^{(1)} \xrightarrow{F} X^{(2)} \xrightarrow{F} \ldots \xrightarrow{F} X^{(b-1)} \xrightarrow{F} C_j.$$

Slide Attacks for Block Ciphers

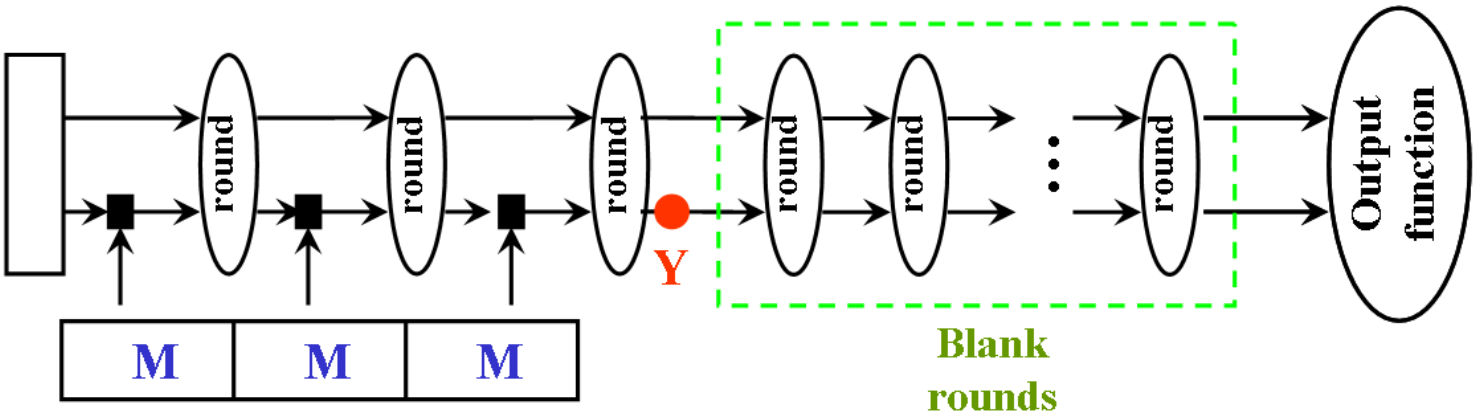
To mount a slide attack one has to find a slid pair of plaintexts (P_i, P_j), such that $P_j = F(P_i)$ and $C_j = F(C_i)$ holds



With the birthday paradox, only $2^{n/2}$ plaintexts are required to find a slid pair

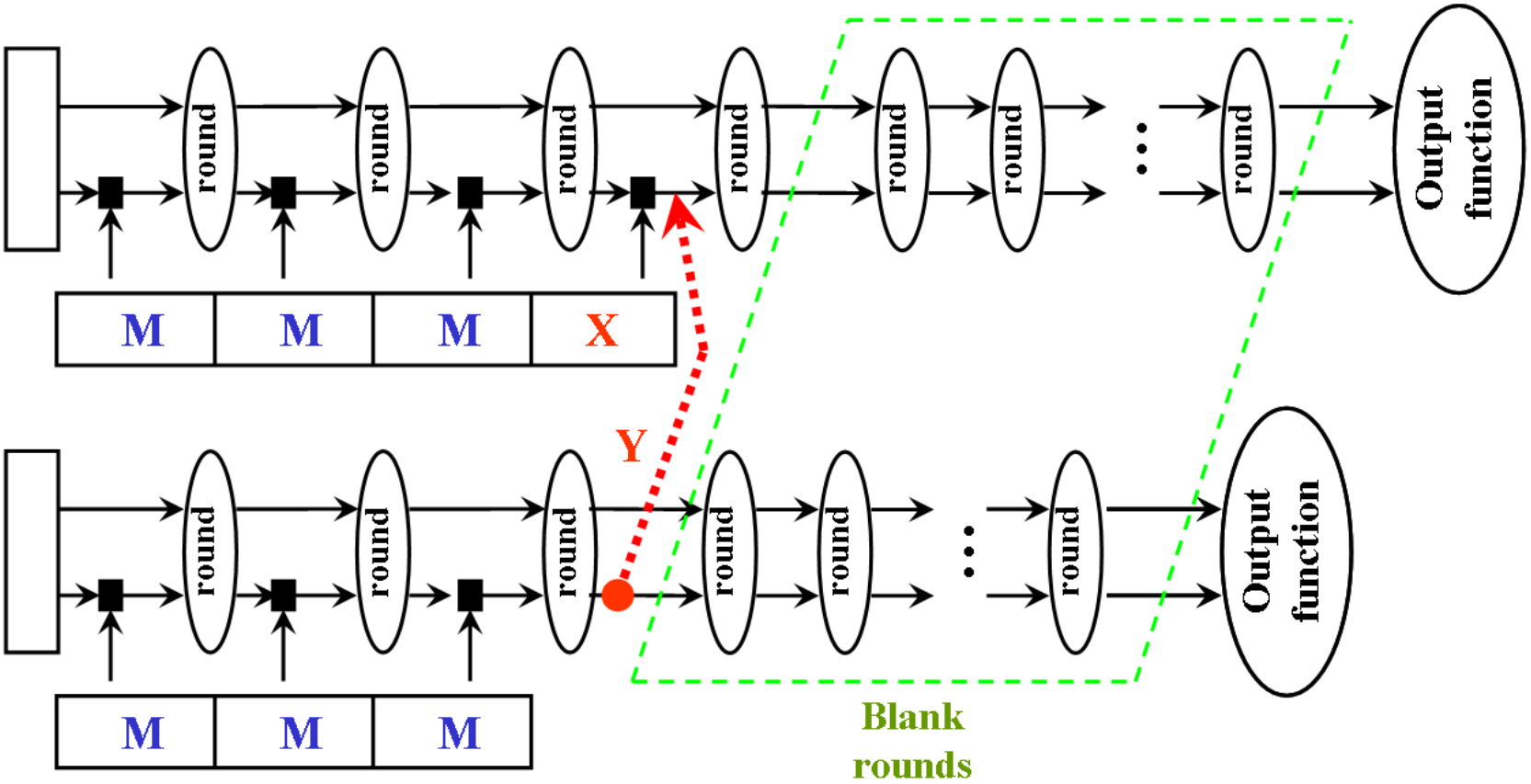
Application of slide attacks against hash functions were very few studied (Saarinen applied slide attacks against the inner cipher of SHA-1)

Slide Attacks on Sponge Functions



Slide Attacks on Sponge Functions

If the addition of X is neutral, then $output1 = round(output2)$



Slide Attacks for Hash Functions

What can we obtain from slide attacks ?

- slide attacks are a typical block cipher cryptanalysis technique
- doesn't seem useful for collision or preimage attacks ...
- ... but we can "distinguish" the hash function from a random oracle
- the key recovery attack may also be useful if some secret is used in the hash function: we can attack a MAC construction using a hash function

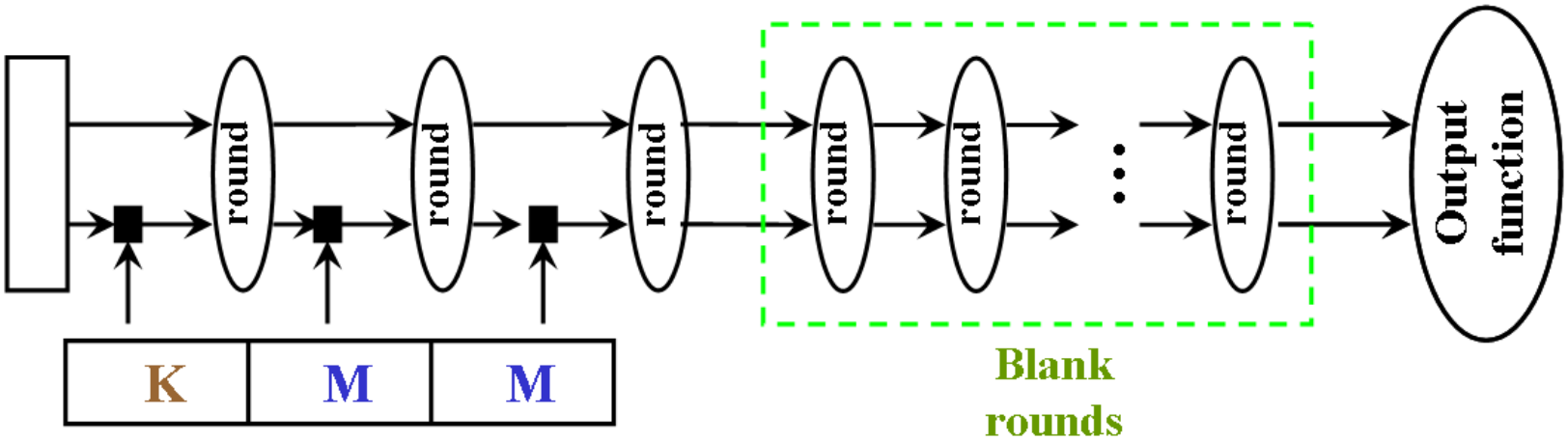
We'll try to attack the following MAC construction:

$$\mathrm{MAC}(K, M) = H(K||M)$$

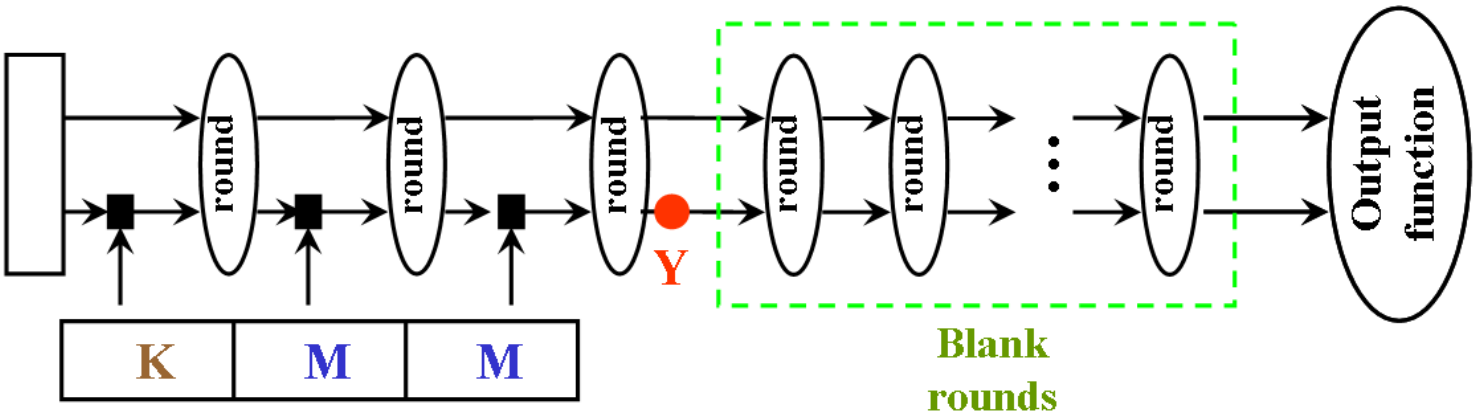
Why Slide Attacks for Sponge Functions

$$\mathrm{MAC}(K, M) = H(K||M)$$

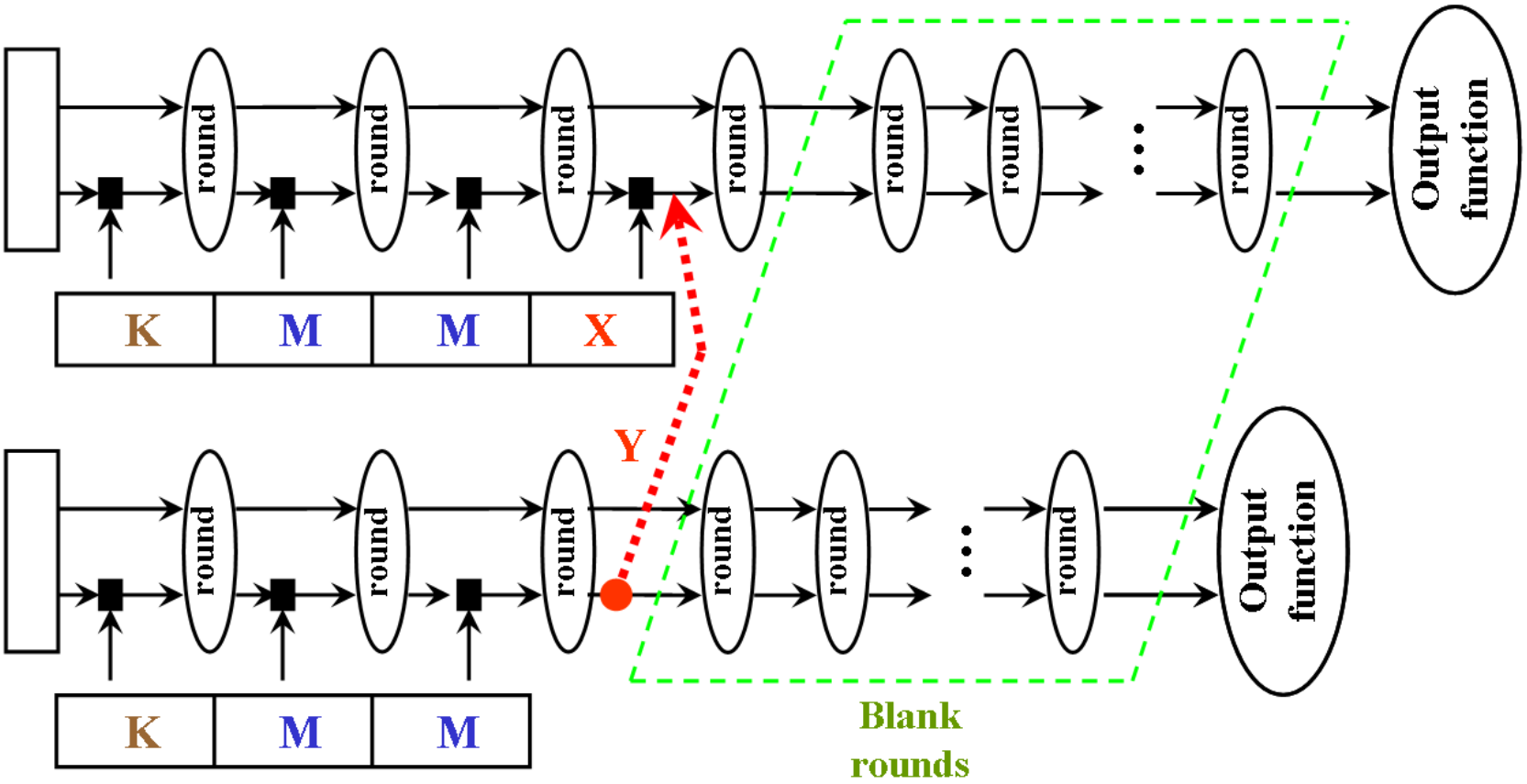
HMAC would be very slow with a sponge function, due to the blank rounds. Thus, the authors advised the following MAC construction:



Slide Attacks on Sponge Functions



Slide Attacks on Sponge Functions



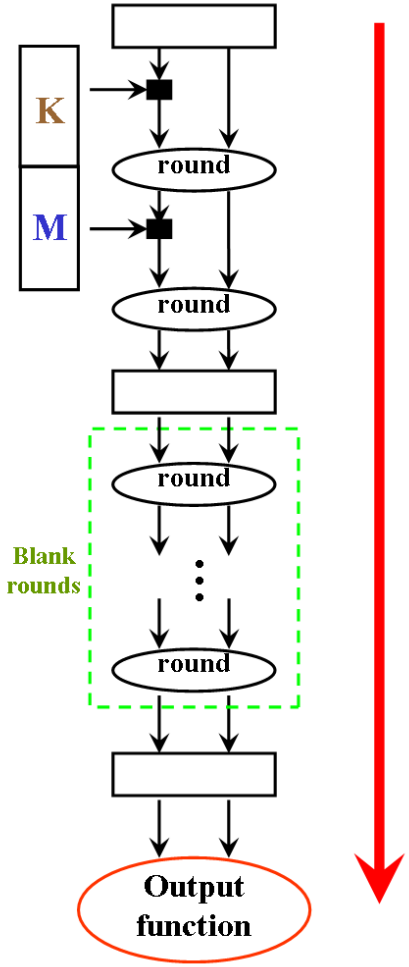
Slide Attacks for Sponge Functions

The Attack Scenario: the attacker makes queries M_i and receive replies $H(K||M_i)$. He then tries to get some non trivial information from the secret K or manage to forge another MAC with good probability.

The attack will be in three steps:

- Find and detect slid pairs of messages
- Recover the internal state
- Uncover some part of the secret key (or forge a new MAC)

The padding must also be taken in account !



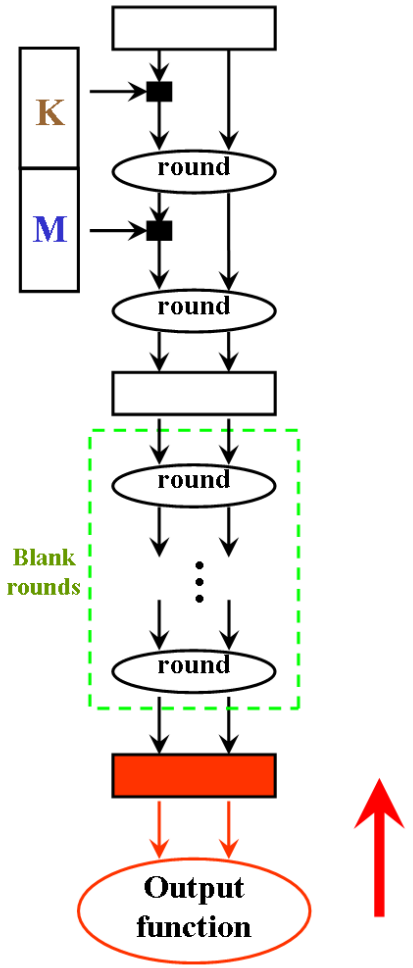
Slide Attacks for Sponge Functions

The Attack Scenario: the attacker makes queries M_i and receive replies $H(K||M_i)$. He then tries to get some non trivial information from the secret K or manage to forge another MAC with good probability.

The attack will be in three steps:

- Find and detect slid pairs of messages
- Recover the internal state
- Uncover some part of the secret key (or forge a new MAC)

The padding must also be taken in account !



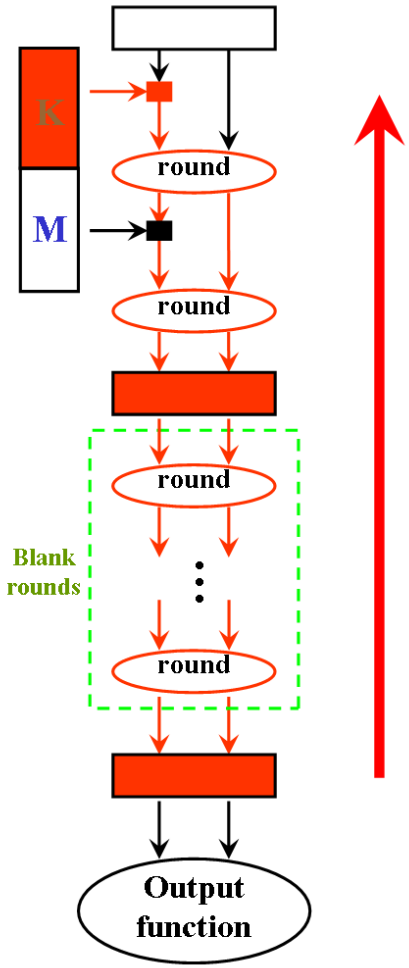
Slide Attacks for Sponge Functions

The Attack Scenario: the attacker makes queries M_i and receive replies $H(K||M_i)$. He then tries to get some non trivial information from the secret K or manage to forge another MAC with good probability.

The attack will be in three steps:

- Find and detect slid pairs of messages
- Recover the internal state
- Uncover some part of the secret key (or forge a new MAC)

The padding must also be taken in account !



Find and detect slid pairs of messages

Find a slid pair of messages:

- depends on the message insertion function
- impossible in the original sponge framework (in which the last inserted word must be different from 0) ...
- ... but possible if a different padding is used !
- possible if the insertion function overwrites the corresponding internal state words (as in GRINDAHL) with $P = 2^{-r}$

Detect a slid pair of messages:

- depends on the output function
- very easy with the sponge squeezing process (all the output words are shifted by one iteration position)
- more complicated with a direct truncation after the blank rounds

Recovering the internal state and **uncovering the secret key** both depend on the whole hash function (require a case by case analysis)

Patches

It is very easy (and costless) for the designers to protect themselves against slide attacks.

If you're inserting message blocks with a XOR:

- just use exactly the sponge framework and make sure that the last inserted message work is different from zero

If you're inserting message blocks by overwriting the corresponding internal state words:

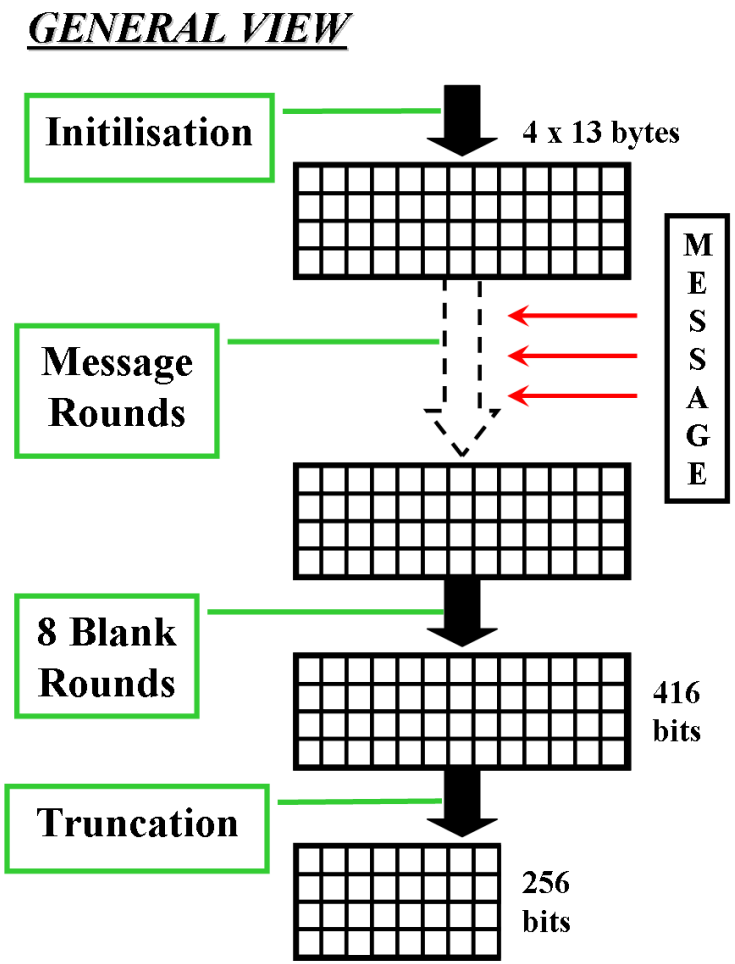
- add a constant to the internal state just before the blank rounds to clearly separate them from the normal rounds
- use a different transformation during the blank rounds

Outline

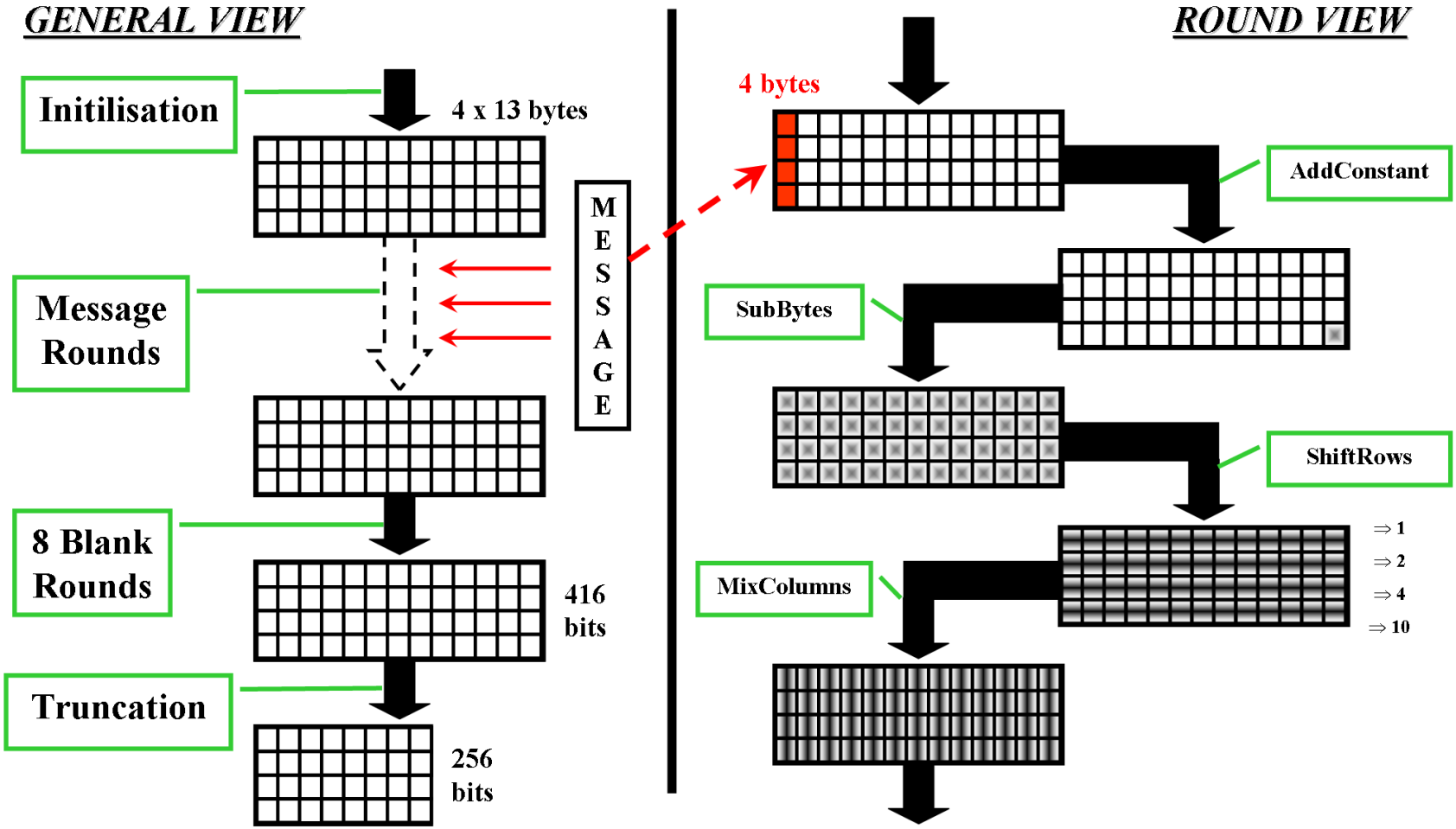
GRINDAHL (Knudsen, Rechberger, Thomsen - 2007)

- 256-bit output (a 512-bit version is also defined)
- fits the framework of extended sponge functions
- **based on AES:** faster than SHA-256 and low memory requirements (can benefit from the fast/small AES implementations)
- collision resistance, 2nd preimage and preimage resistance in $2^{n/2}$ function calls (possibility of meet-in-the-middle attacks for (2nd)-preimage)

GRINDAHL (Knudsen, Rechberger, Thomsen - 2007)



GRINDAHL (Knudsen, Rechberger, Thomsen - 2007)



The padding of GRINDAHL

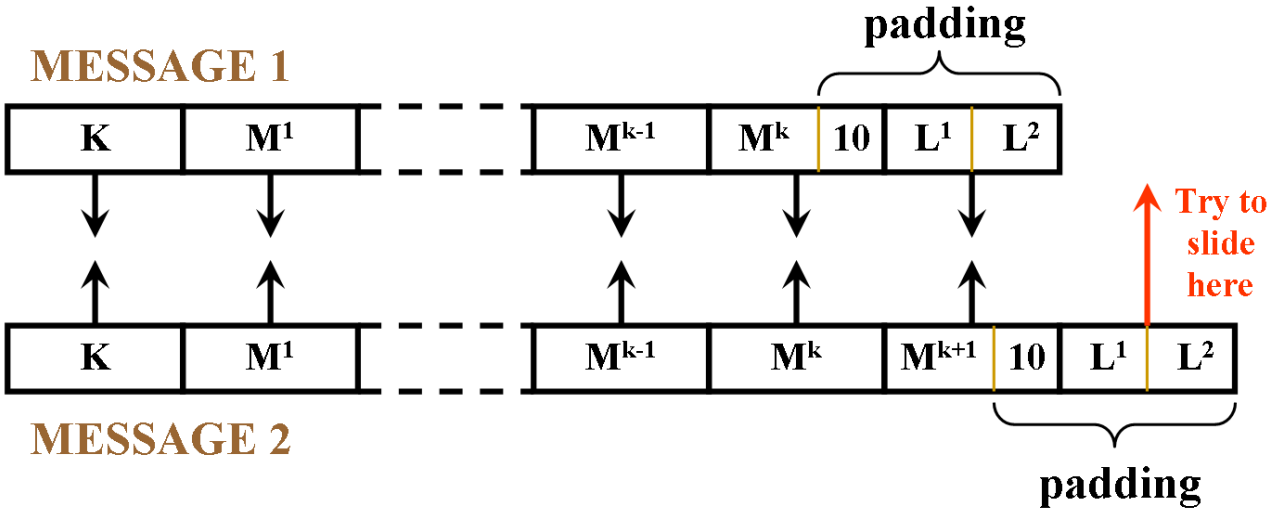
In GRINDAHL we have 10-padding and length-padding:

- **10-padding** appends a “1”-bit to the message, followed by as many “0”-bits as needed to complete the last message block
- **length-padding** then appends the number of message blocks (not bits!) for the entire padded message as a 64-bit value (two blocks of message for GRINDAHL-256, one for GRINDAHL-512)

One effect of the 10-padding is that the last message block before the length-padding can be any value, except for the all-zero block

Finding slid pairs for GRINDAHL

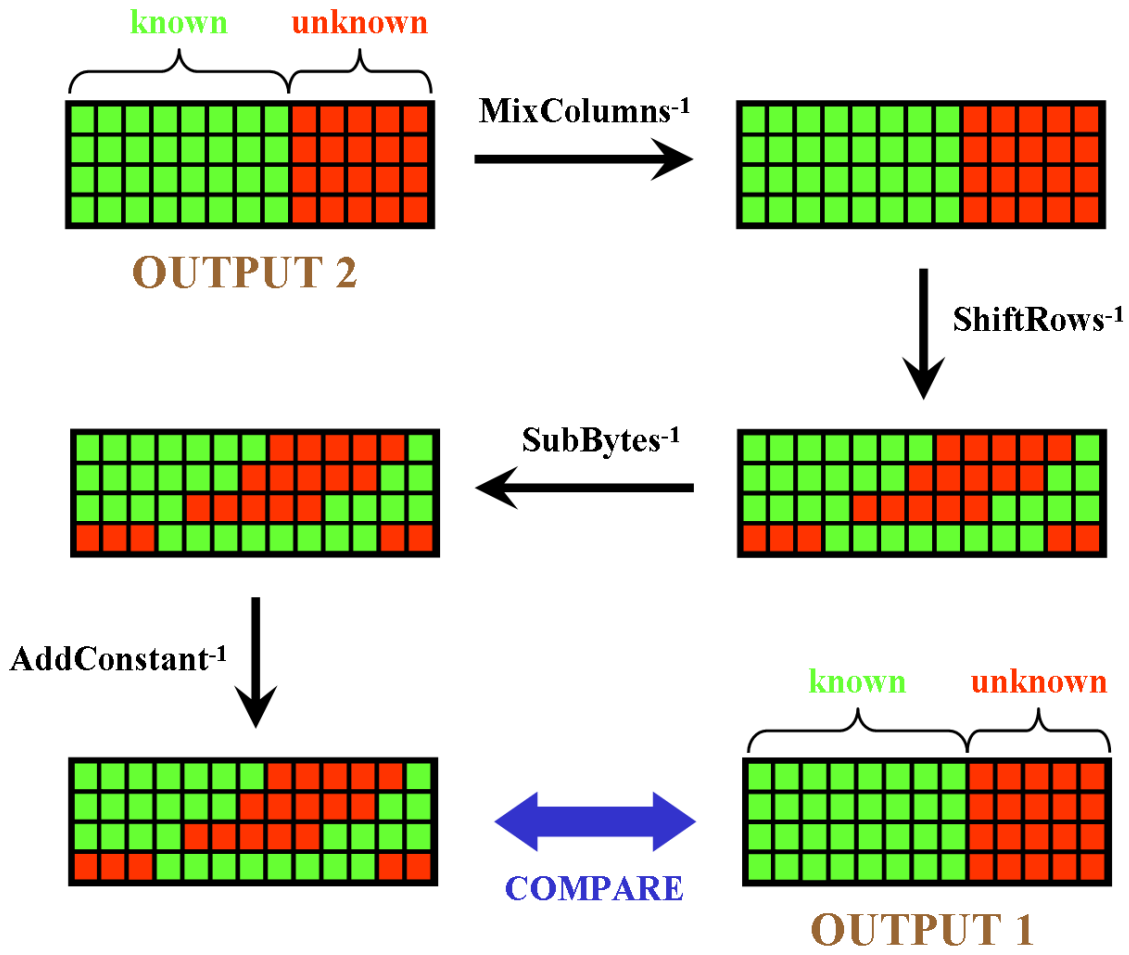
Suppose the length padding fits in one message block
(true for 512-bit version, a little bit more complicated technique is required for the 256 version)



We have a probability of $2^{-r} = 2^{-32}$ to get a slid pair (for 512-bit version $P = 2^{-64}$)

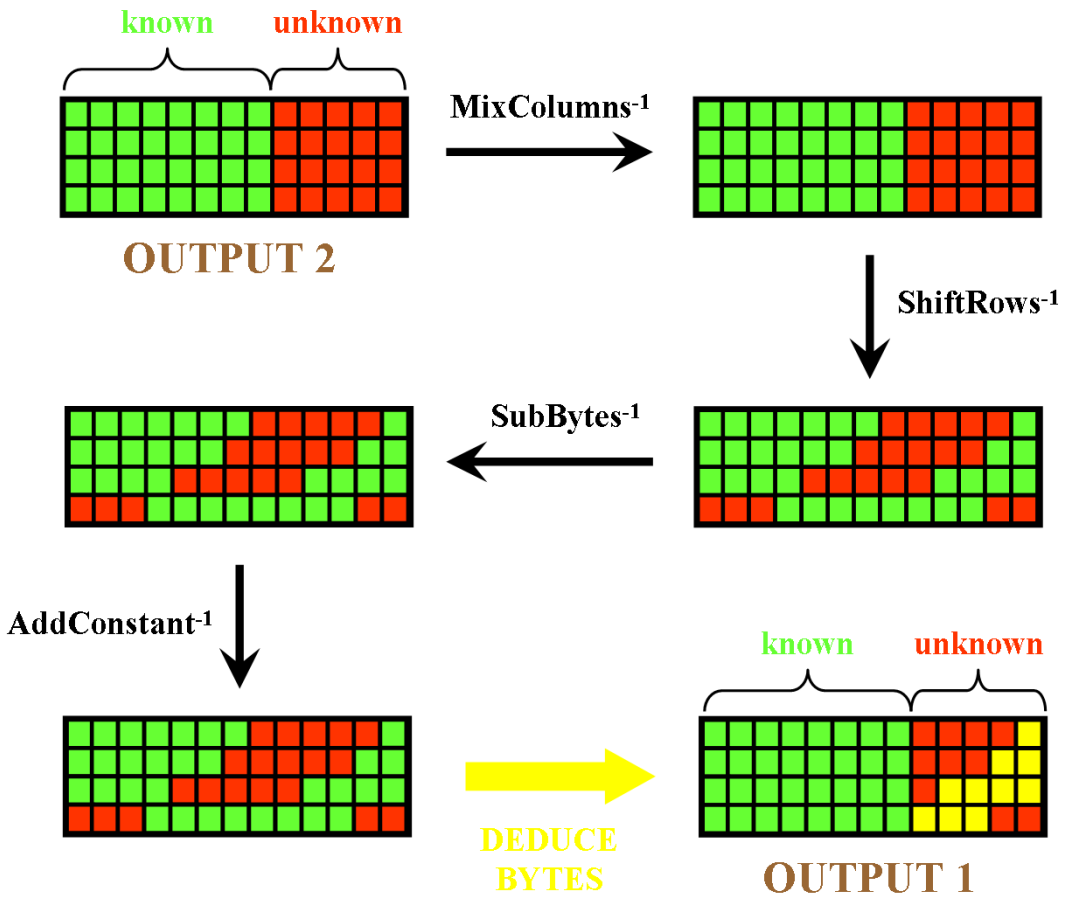
Detecting slid pairs for GRINDAHL

Assume two messages M_1 and M_2 that are slid. **How to detect them, just by looking at the corresponding outputs ?**



Recovering the internal state for GRINDAHL

Assume we found and detected a slid pair (M_1, M_2), **we already know the truncated internal state from M_1 and we'll try to recover the rest thanks to the truncated output of M_2**



Once the internal state before the truncation is fully recovered, **one can completely invert the blank rounds**

More results

For GRINDAHL**-256**, the attack allows to:

- distinguish from RO with 2^{64} queries and computation time
- forge valid MACs or to recover 1 new byte of the secret with 2^{64} queries and 2^{80} computations

For GRINDAHL**-512:** the attack allows to (**first cryptanalytic results on this version**):

- distinguish from RO with 2^{64} queries and computation time
- forge valid MACs or to recover 4 new bytes of the secret with 2^{64} queries and 2^{80} computations

For RADIOGATÚN**: attack don't apply**, but would work on an overwrite version of it

Outline

Properties of GRINDAHL

Main security arguments:

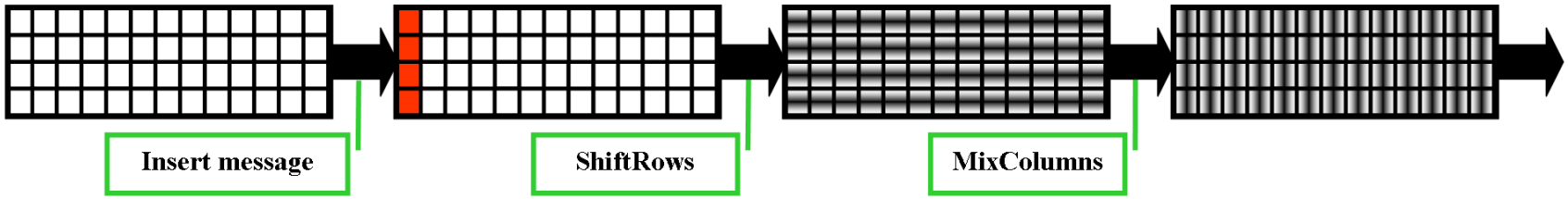
- a collision requires intermediate states with at least half of the bytes active
- an internal collision requires at least 5 rounds

It is very hard to find a low-weight and-or a small differential path for GRINDAHL

Truncated differentials

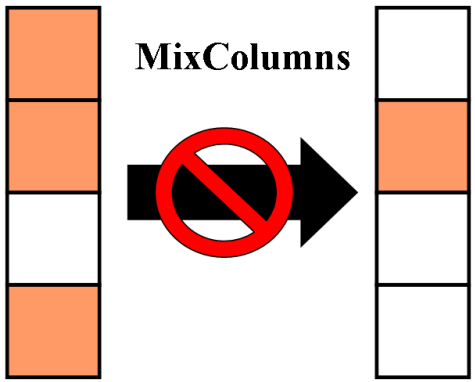
- the scheme is byte oriented
- let's deal with **truncated differences**: only check if there is a difference in a byte, but don't care about the actual value of the difference
- we can forget about SubBytes and the constant addition (transparent for truncated differentials)
- *we only deal with ShiftRows, MixColumns and truncation*

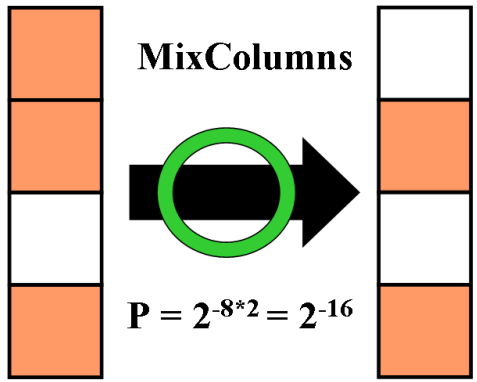
The simplified scheme we consider:



The MixColumns function

- How do the truncated differentials react with the MixColumns function ?
- **Property of MixColumns:**
  $\sharp\{\text{input byte-differences}\} + \sharp\{\text{output byte-differences}\} \geq 5$
- **P[valid transitions] =** $2^{-8\times(4-\sharp\{\text{output byte-differences}\})}$.



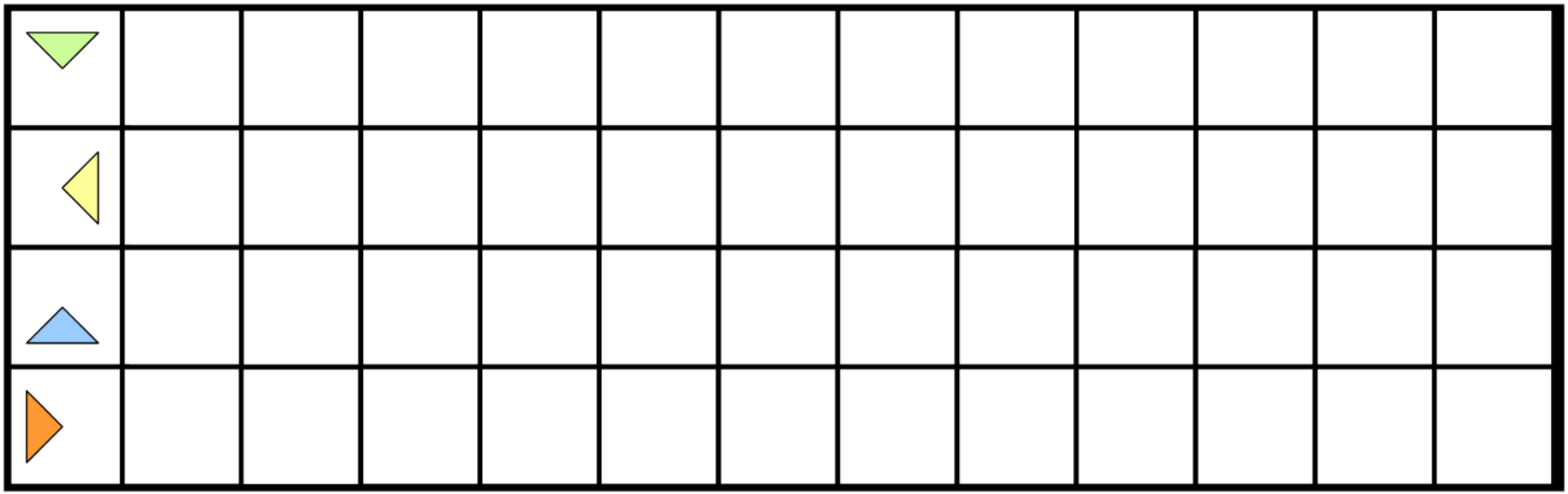


The control bytes (1)

- ShiftRows modified (1, 2, 4, 10) for better diffusion: every state byte depends on every message byte after 4 rounds
- ... but what happens before those 4 rounds ?
- each message byte inserted affect some subset of the internal state S
- **this will allow us to control a little bit the difference spreading by forcing some MixColumns differential transitions independently**
- we call them **control bytes**

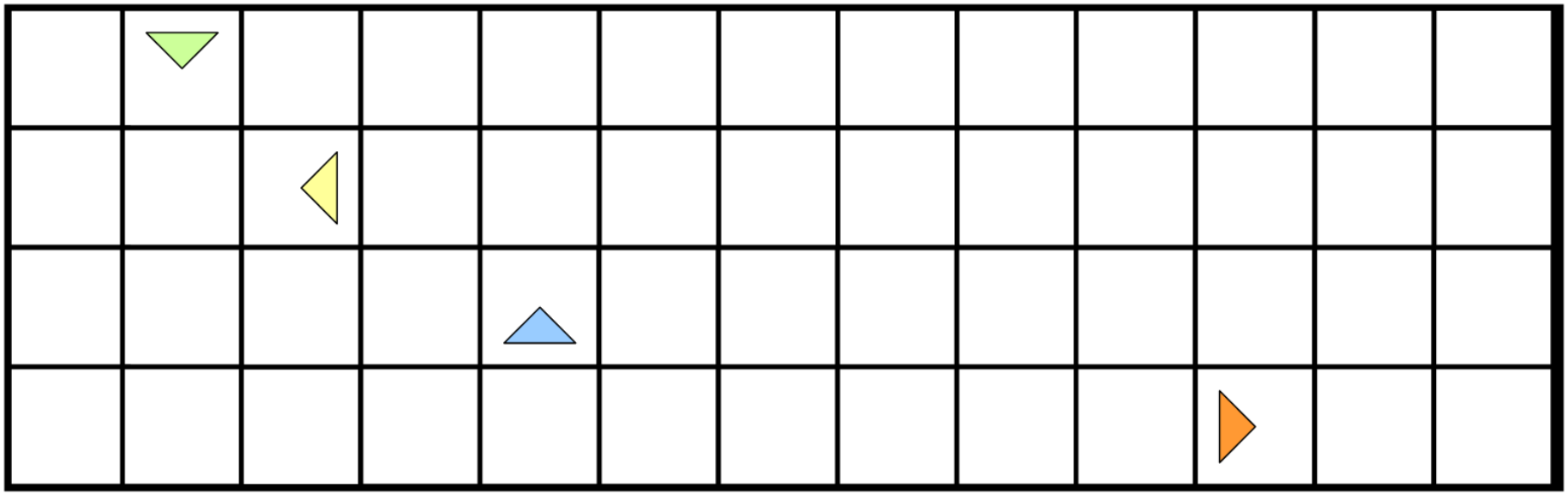
The control bytes (2)

- **Insert the message bytes**



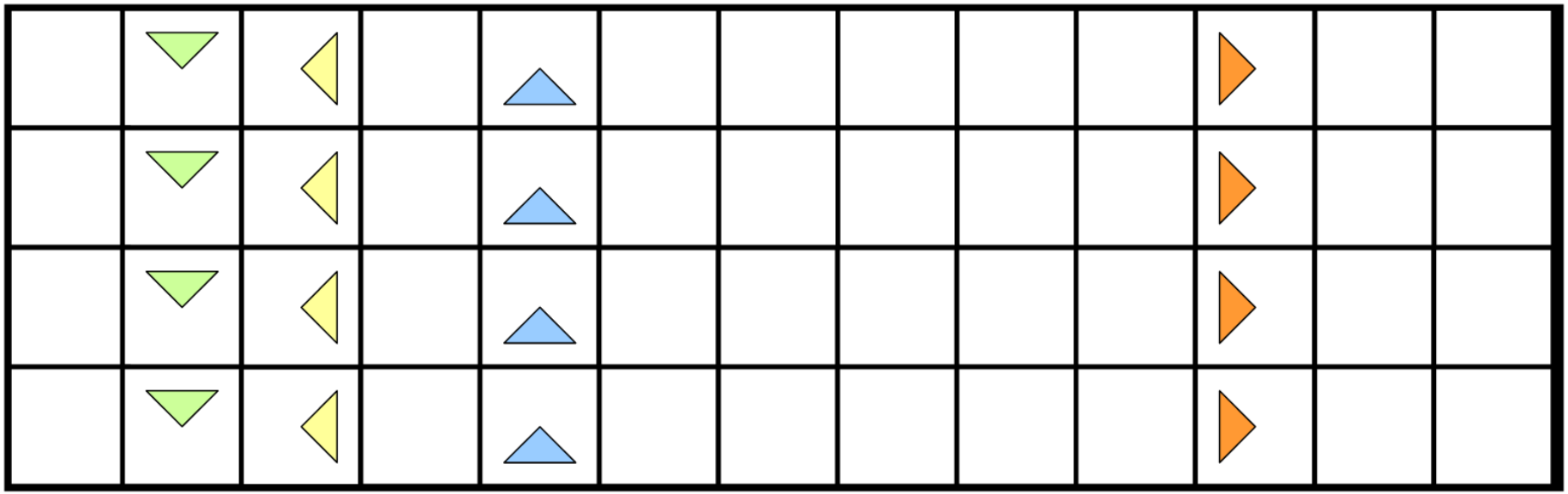
The control bytes (2)

- Do **ShiftRows** (1st round)



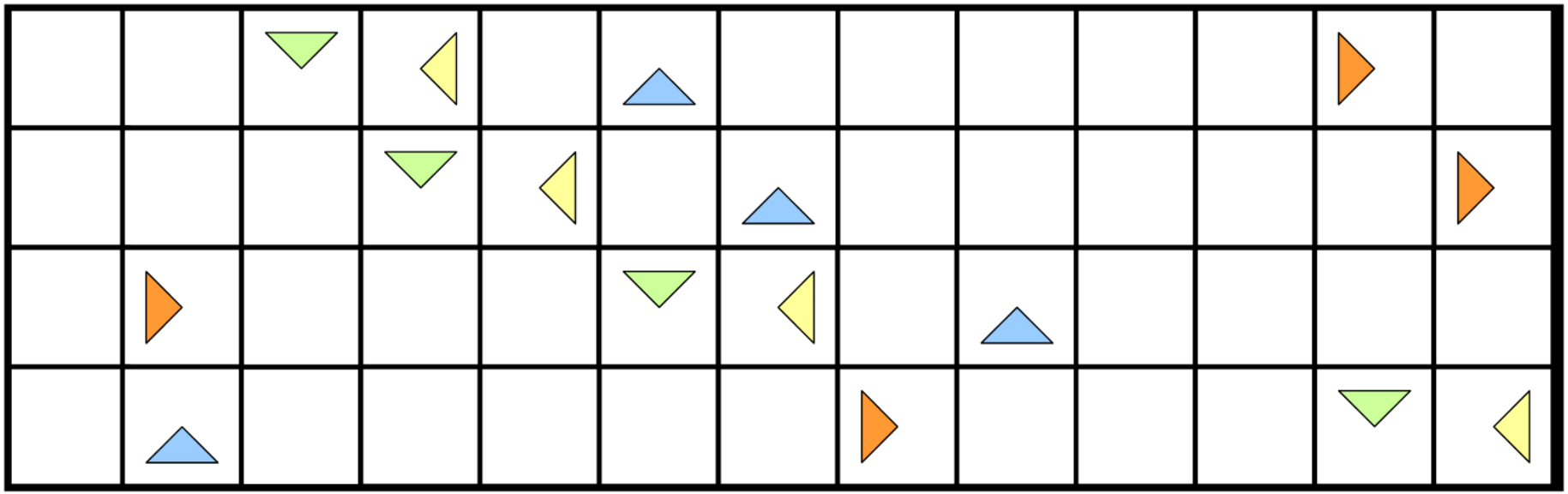
The control bytes (2)

- Do **MixColumns** (1^{st} round)



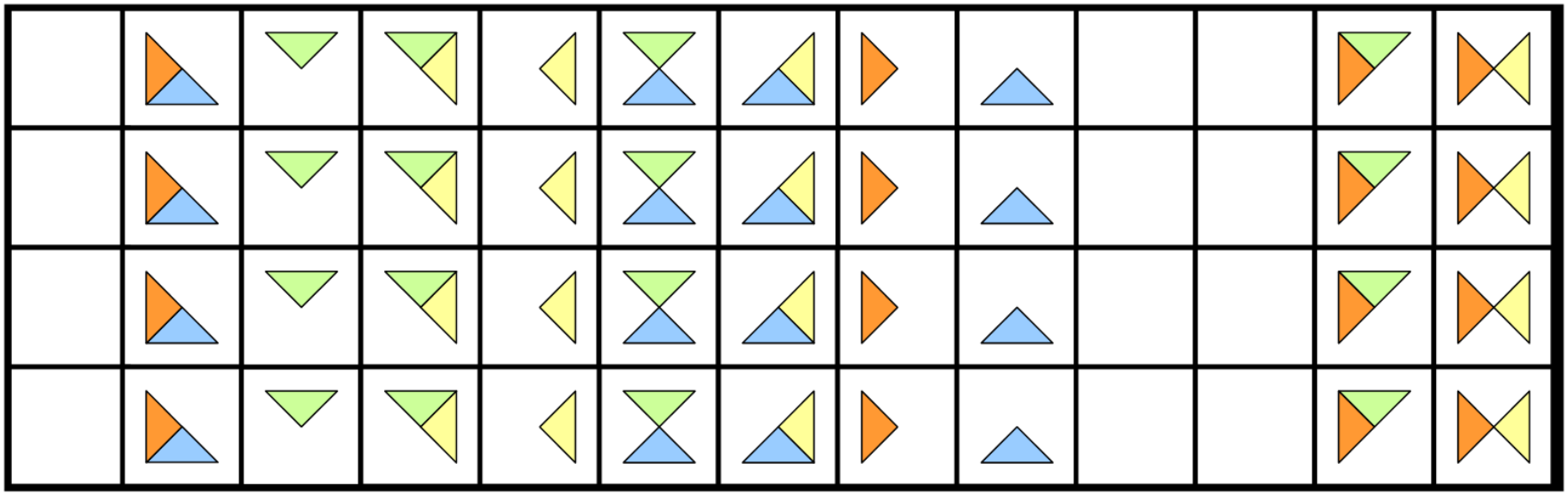
The control bytes (2)

- Do **ShiftRows** (2^{nd} round)



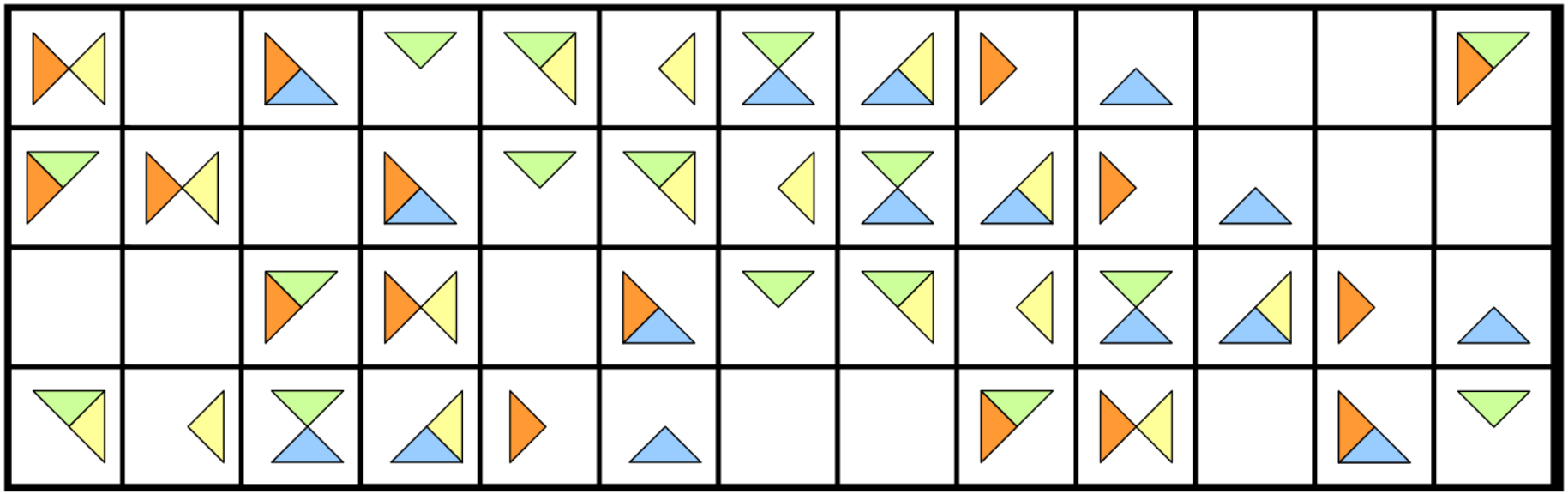
The control bytes (2)

- Do **MixColumns** (2^{nd} round)



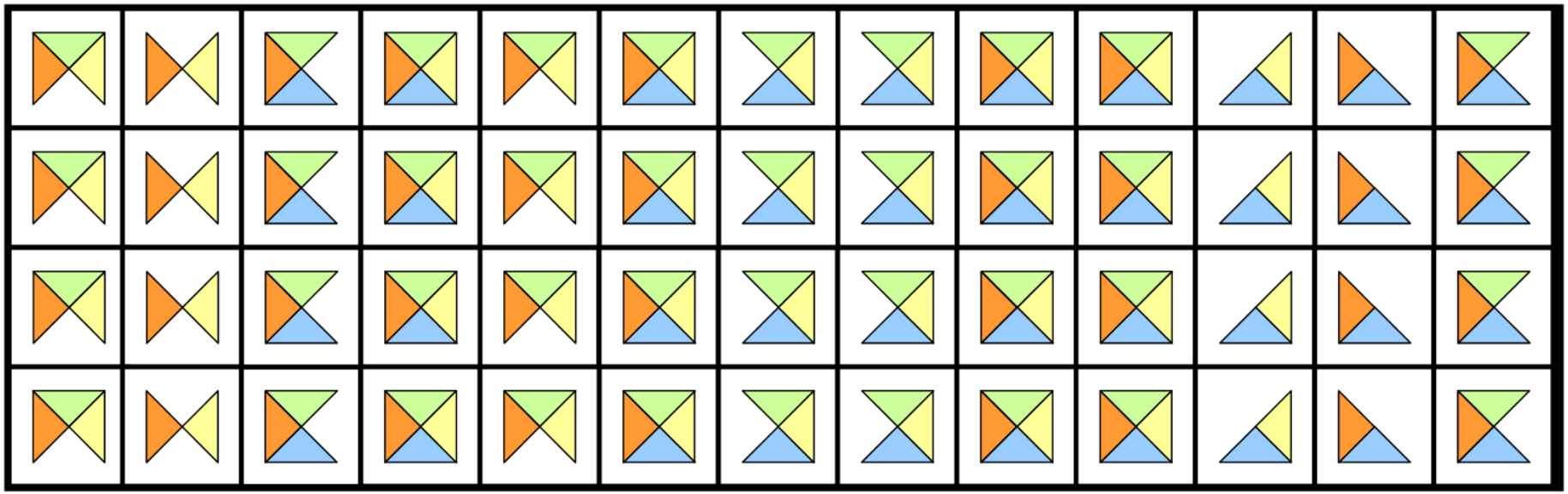
The control bytes (2)

- Do **ShiftRows** (3^{rd} round)



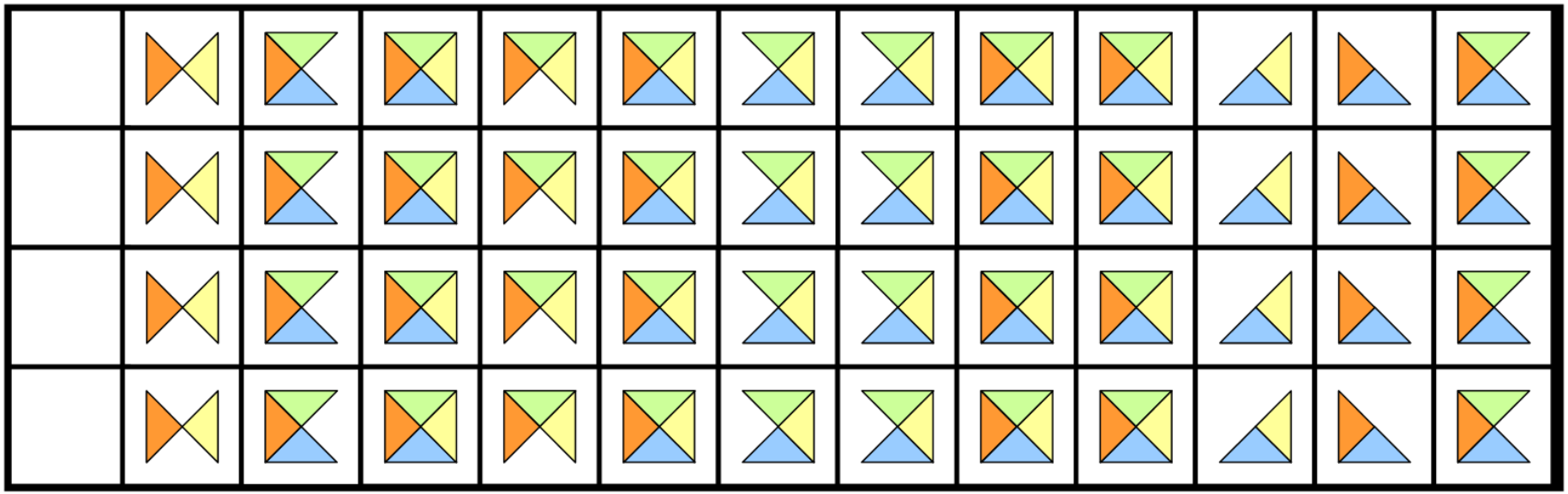
The control bytes (2)

- Do **MixColumns** (3^{rd} round)



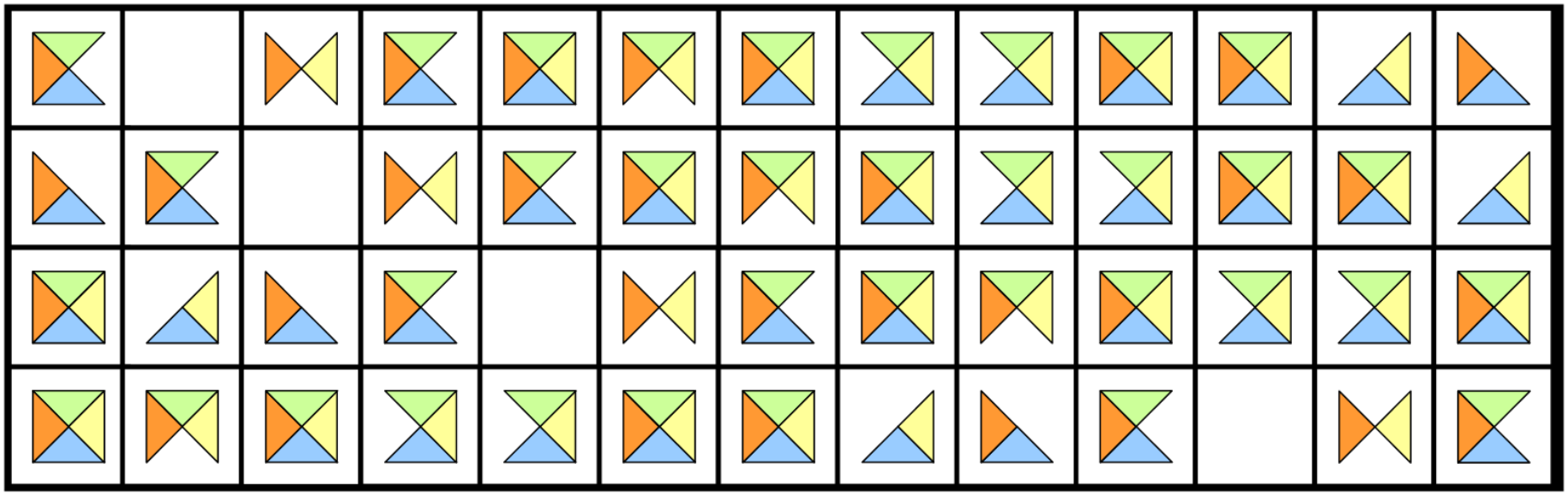
The control bytes (2)

- **Truncation of the first column** (new message bytes)



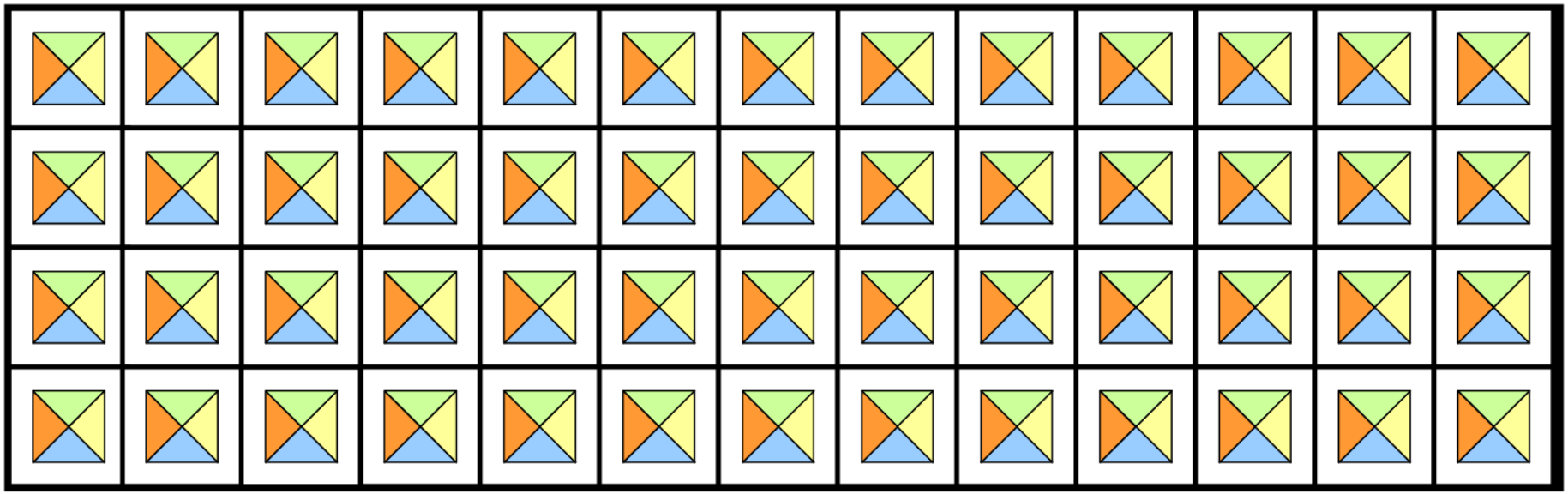
The control bytes (2)

- Do **ShiftRows** (4^{th} round)



The control bytes (2)

- Do **MixColumns** (4^{th} round)



Internal collisions are better

- 2 possibilities for a collision: internal or not
- the blank rounds would make things really hard since we have no more control (no more message byte inserted)
- an **internal collision** seems easier, even if we can not use the final truncation anymore (we'll have a bigger internal state to make collide)
- **2 possibles ways to erase a truncated difference**: with a MixColumns transition (for a cost P^{-1}) or thanks to the overwriting during a message insertion (no cost since already planed in the differential path)

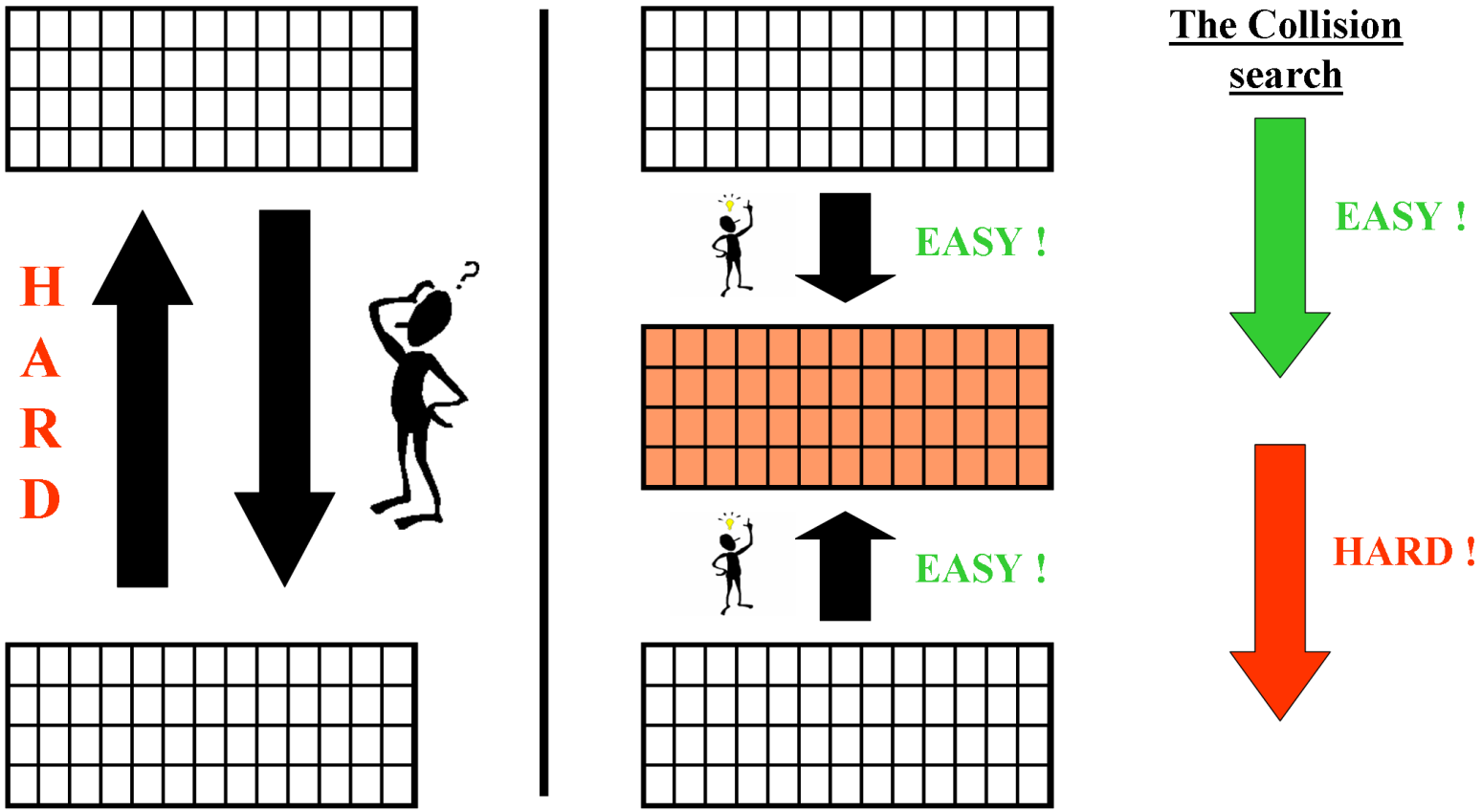
An unintuitive strategy

- Building a differential path is really hard because of the two security properties
- **idea - take the all-difference state as a check point:**
  - from a no-difference state to an all-difference state: hopefully very easy ! No need for a differential path here
  - from an all-difference state to a no-difference state: harder ! Build the differential path backward and search for a collision onward
- the costly part is obviously the second stage !

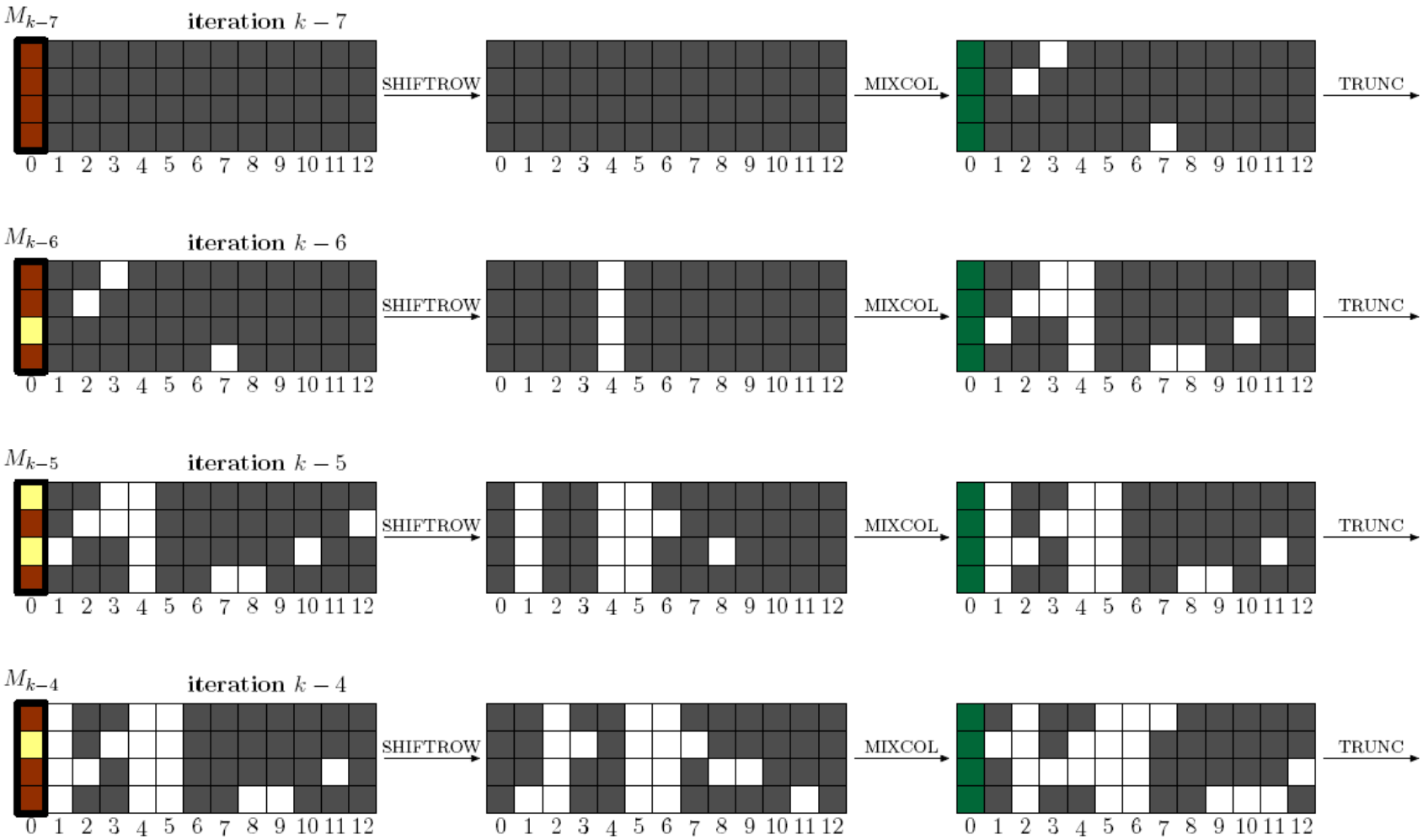
That is an unintuitive strategy for a hash function cryptanalyst: we deliberately let all the differences spread in the whole state before beginning the collision search !

How to build a differential path

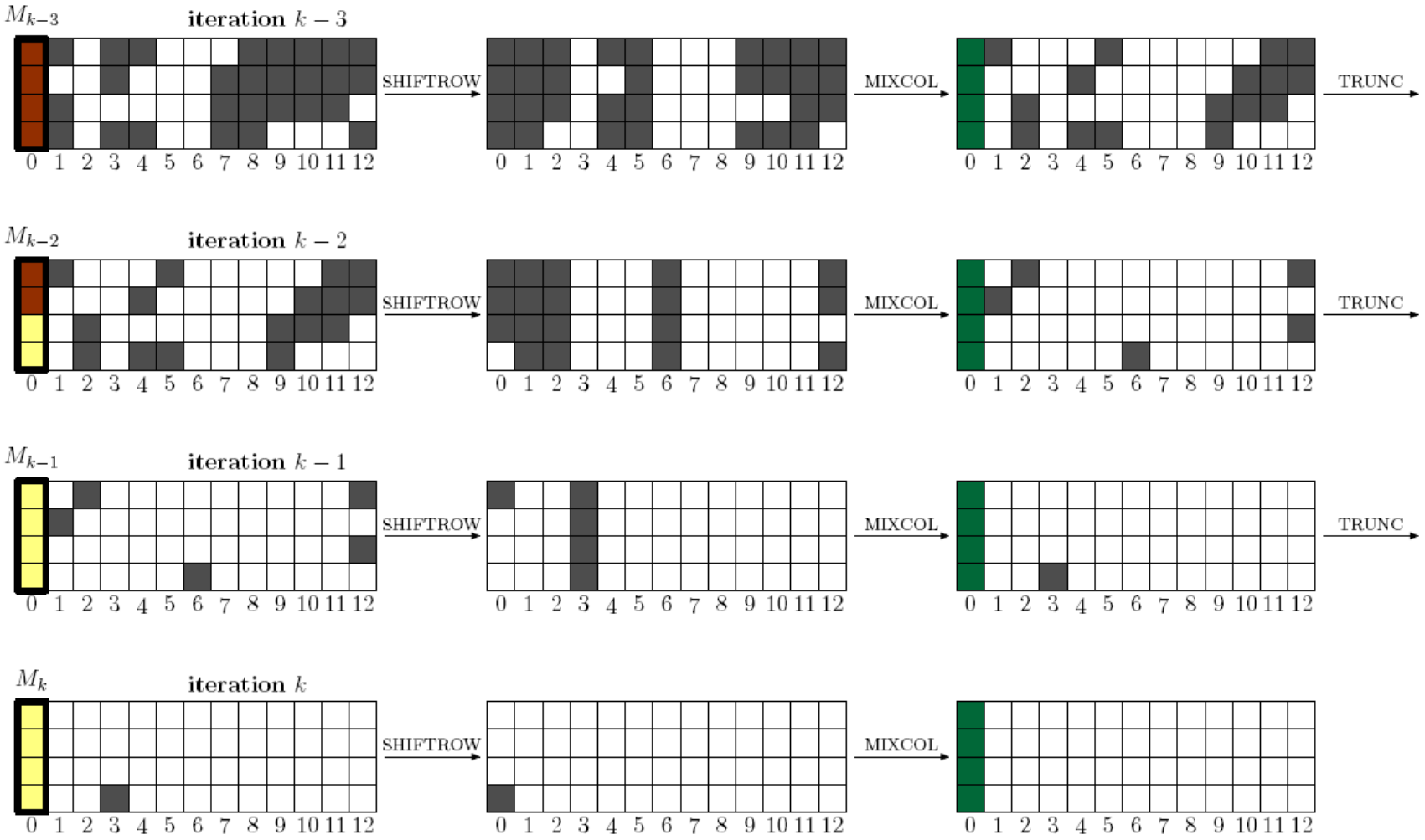
Building a differential path is really hard !



Our truncated differential path (1)



Our truncated differential path (1)



The collision attack

The attack is in three steps:

- 1^{st} **step:** reach an all-difference state (for example by adding a lot of differences very quickly) and generate $K = 2^{112}$ other all-difference states from it
  - P[all-difference state to all-difference state]$\simeq 2^{-0,27}$
- 2^{nd} **step:** for each all-difference state, check if one can find a message pair following the differential path
  - P[without control bytes]=2^{-440}
  - P[with control bytes]=2^{-112}
- 3^{rd} **step:** once a valid message pair found, add a random message block without difference in order to force the first column overwriting in the last step

Any question ?