Twenty Fifth CSF 2012

# Verified Security of Merkle-Damgård

Michael Backes[1,2] Gilles Barthe[3] **Matthias Berg**[1]
Benjamin Grégoire[4] César Kunz[5,3] Malte Skoruppa[1]
Santiago Zanella Béguelin[6]

[1]Saarland University
[2]MPI-SWS
[3]IMDEA Software Institute
[4]INRIA Sophia Antipolis-Méditerranée
[5]Universidad Politécnica de Madrid
[6]Microsoft Research

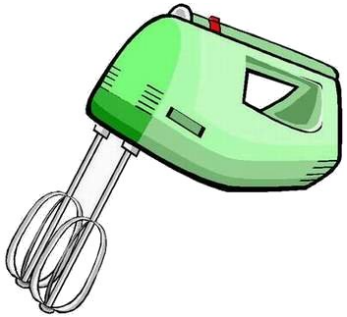
June 27, 2012


SHA-1
32oz
30oz
28oz
26oz
24oz
22oz
20oz
18oz
16oz
14oz
12oz
10oz
8oz
6oz
4oz
2oz
NSF

# Hash security

- November 2007: SHA-3 competition launched
- *"The security provided by an algorithm is the most important factor in the evaluation."* [NIST]
  - Minimum requirements: collision resistance, preimage resistance, second-preimage resistance, resistance to length-extension attacks
- December 2010: 5 finalists

BLAKE



Grøstl

JH

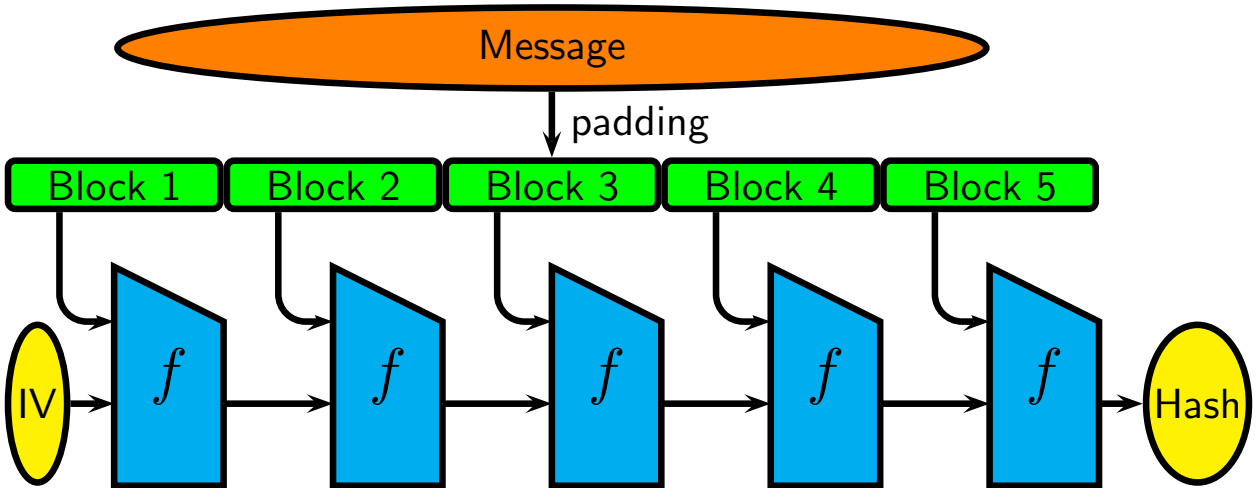
Keccak

Skein

## Winner to be chosen in 2012!

# Hash security

- November 2007: SHA-3 competition launched
- *"The security provided by an algorithm is the most important factor in the evaluation."* [NIST]
  - Minimum requirements: collision resistance, preimage resistance, second-preimage resistance, resistance to length-extension attacks
- December 2010: 5 finalists
  - All 5 are variants of the Merkle-Damgård construction

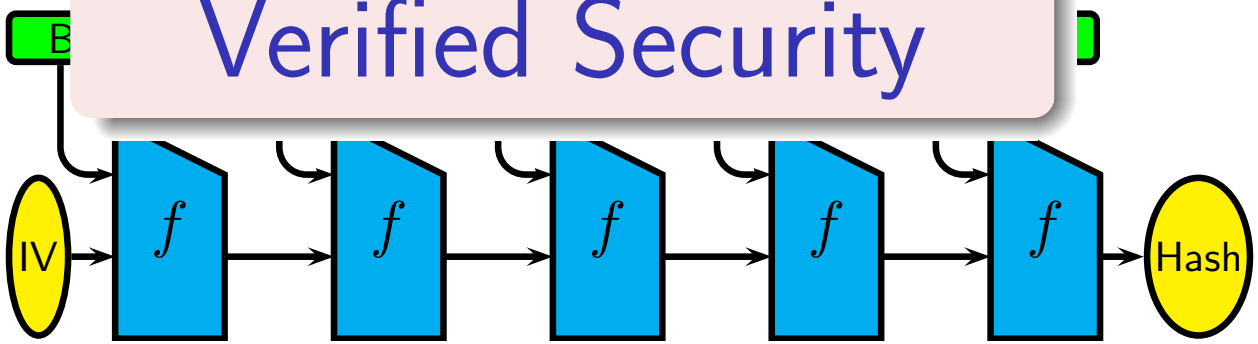


# Hash security

- November 2007: SHA-3 competition launched
- *"The security provided by an algorithm is the most important factor in the evaluation."* [NIST]
  - Minimum requirements: collision resistance, preimage resistance, second-preimage resistance, resistance to length-extension attacks
- December 2010: 5 finalists
  - All 5 are variants of the Merkle-Damgård construction

**Our contribution**

**Formally Verified Security**



# Verified security

## EasyCrypt

- Formal framework for security proofs
- Uses state-of-the-art verification tools


- Makes verified security accessible to non-experts in formal methods
- Supports game-based proofs

$$\boxed{\begin{array}{l}\textbf{Game } \mathsf{G}_1: \\ \ldots \\ \ldots \leftarrow \mathcal{A}(); \\ \ldots\end{array}} \quad \boxed{\begin{array}{l}\textbf{Game } \mathsf{G}_2: \\ \ldots \\ \ldots \leftarrow \mathcal{A}(); \\ \ldots\end{array}} \quad \ldots \quad \boxed{\begin{array}{l}\textbf{Game } \mathsf{G}_n: \\ \ldots \\ \ldots \leftarrow \mathcal{A}(); \\ \ldots\end{array}}$$

$$\Pr[\mathsf{G}_1 : b] \leq \Pr[\mathsf{G}_2 : b] \leq \ldots \leq \Pr[\mathsf{G}_n : b]$$

  - pWhile: typed probabilistic imperative Language
  - probabilistic relational Hoare logic: $\vdash c_1 \sim c_2 : \Psi \Rightarrow \Phi$
  - random sampling, lazy/eager sampling, **loops**, **failure events**, ...

# Verified security

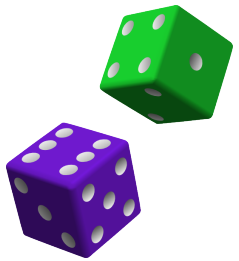
## EasyCrypt

- Formal framework for security proofs
- Uses state-of-the-art verification tools


- Makes verified security accessible to non-experts in formal methods
- Supports game-based proofs

$$\boxed{\textbf{Game } \mathsf{G}_1 : \ldots \ldots \leftarrow \mathcal{A}(); \ldots} \quad \boxed{\textbf{Game } \mathsf{G}_2 : \ldots \ldots \leftarrow \mathcal{A}(); \ldots} \quad \cdots \quad \boxed{\textbf{Game } \mathsf{G}_n : \ldots \ldots \leftarrow \mathcal{A}(); \ldots}$$

$$\Pr[\mathsf{G}_1 : b] \leq \Pr[\mathsf{G}_2 : b] \leq \ldots \leq \Pr[\mathsf{G}_n : b]$$

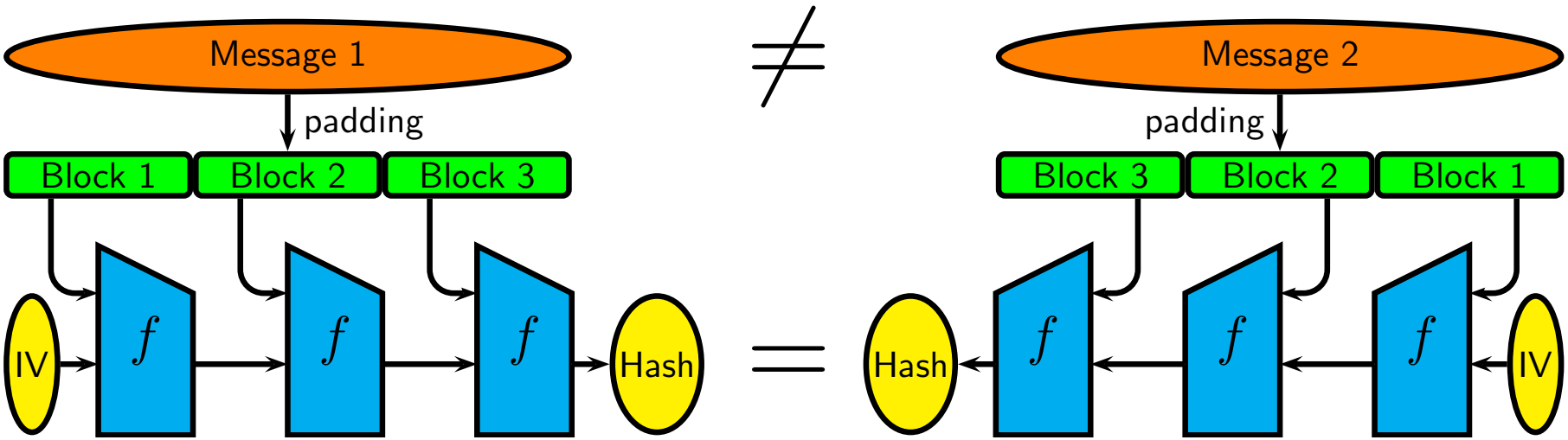


$\equiv$

$$\boxed{\begin{array}{l}\textbf{Oracle } \mathcal{O}(x) : \\ \text{if } x \notin \mathsf{dom}(\boldsymbol{L}) \text{ then} \\ \quad \boldsymbol{L}[x] \xleftarrow{\$} \{0,1\}^n \\ \text{return } \boldsymbol{L}[x]\end{array}}$$

- pWhile: typed probabilistic imperative Language
- probabilistic relational Hoare logic: $\vdash c_1 \sim c_2 : \Psi \Rightarrow \Phi$
- random sampling, lazy/eager sampling, **loops**, **failure events**, ...

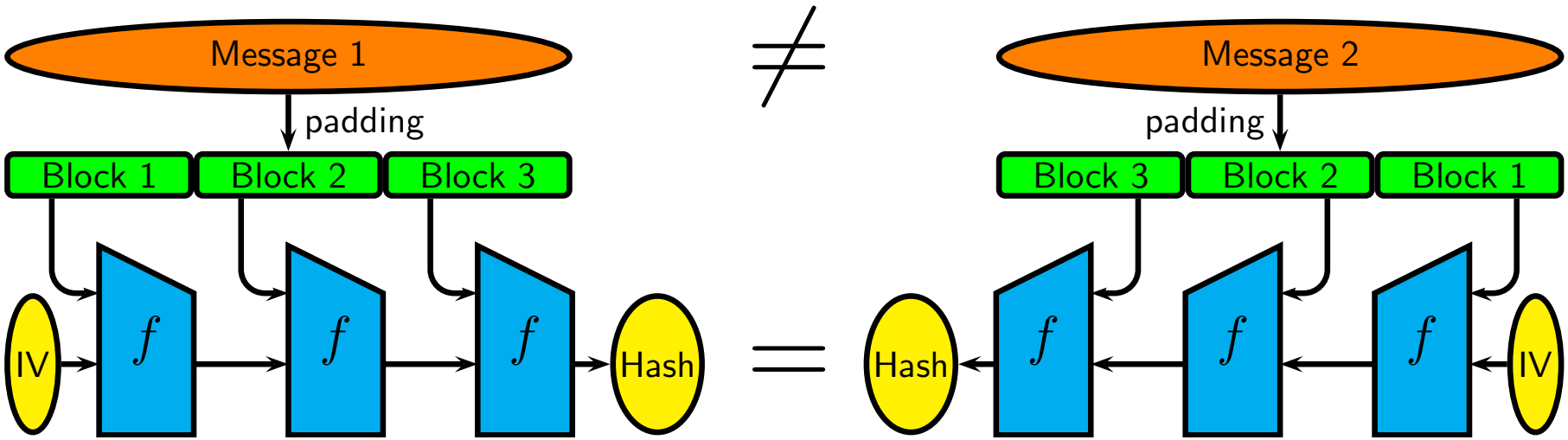
# Collision resistance



# Collision resistance

## Collision resistance [Merkle, Damgård, 1989]

For every collision finder $\mathcal{A}$ for the *suffix-free* Merkle-Damgård construction there exists a collision finder $\mathcal{B}$ for its compression function $f$.



# Collision resistance in EasyCrypt

# Collision resistance in EasyCrypt

- define formal context (types, constants, operators, axioms+lemmas)

op is_coll$(p_1, p_2)$ $\equiv$ $p_1 \neq p_2 \wedge f(p_1) = f(p_2)$

axiom suffix_free: $\forall(m, m', bl), m \neq m' \Rightarrow \mathsf{pad}(m) \neq bl \| \mathsf{pad}(m')$

# Collision resistance in EasyCrypt

- ✓ define formal context (types, constants, operators, axioms+lemmas)

# Collision resistance in EasyCrypt

- ✓ define formal context (types, constants, operators, axioms+lemmas)
- define games (variables, procedures, adversaries)

**Game** $\mathsf{CR}^{\mathsf{MD}}$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
$h_1 \leftarrow \mathsf{MD}(m_1);$
$h_2 \leftarrow \mathsf{MD}(m_2);$
return $(m_1 \neq m_2 \wedge h_1 = h_2)$

**Game** $\mathsf{CR}^{f}$ :
$(p_1, p_2) \leftarrow \mathcal{B}();$
return $\mathsf{is\_coll}(p_1, p_2)$

**Adversary** $\mathcal{B}()$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
[. . . *code omitted*. . . ]
return $(p_1, p_2)$

# Collision resistance in EasyCrypt

- ✓ define formal context (types, constants, operators, axioms+lemmas)
- ✓ define games (variables, procedures, adversaries)

**Game** $\mathsf{CR}^{\mathsf{MD}}$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
$h_1 \leftarrow \mathsf{MD}(m_1);$
$h_2 \leftarrow \mathsf{MD}(m_2);$
return $(m_1 \neq m_2 \wedge h_1 = h_2)$

**Game** $\mathsf{CR}^{f}$ :
$(p_1, p_2) \leftarrow \mathcal{B}();$
return $\mathsf{is\_coll}(p_1, p_2)$

**Adversary** $\mathcal{B}()$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
[. . . *code omitted*. . . ]
return $(p_1, p_2)$

- prove logical judgments to establish equivalences between games

$$\vdash \mathsf{CR}^{\mathsf{MD}} \sim \mathsf{CR}^{f} : \mathsf{true} \Rightarrow \mathsf{res}\langle 1 \rangle \rightarrow \mathsf{res}\langle 2 \rangle$$

# Collision resistance in EasyCrypt

- ✓ define formal context (types, constants, operators, axioms+lemmas)
- ✓ define games (variables, procedures, adversaries)

**Game** $\mathsf{CR}^{\mathsf{MD}}$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
$h_1 \leftarrow \mathsf{MD}(m_1);$
$h_2 \leftarrow \mathsf{MD}(m_2);$
return $(m_1 \neq m_2 \wedge h_1 = h_2)$

**Game** $\mathsf{CR}^{f}$ :
$(p_1, p_2) \leftarrow \mathcal{B}();$
return $\mathsf{is\_coll}(p_1, p_2)$

**Adversary** $\mathcal{B}()$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
[. . . *code omitted*. . . ]
return $(p_1, p_2)$

- ✓ prove logical judgments to establish equivalences between games

$$\vdash \mathsf{CR}^{\mathsf{MD}} \sim \mathsf{CR}^{f} : \mathsf{true} \Rightarrow \mathsf{res}\langle 1 \rangle \rightarrow \mathsf{res}\langle 2 \rangle$$

- derive inequalities between probabilities of events in games

$$\Pr\left[\mathsf{CR}^{\mathsf{MD}} : \mathsf{res}\right] \leq \Pr\left[\mathsf{CR}^{f} : \mathsf{res}\right]$$

# Collision resistance in EasyCrypt

- ✓ define formal context (types, constants, operators, axioms+lemmas)
- ✓ define games (variables, procedures, adversaries)

**Game** $\mathsf{CR}^{\mathsf{MD}}$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
$h_1 \leftarrow \mathsf{MD}(m_1);$
$h_2 \leftarrow \mathsf{MD}(m_2);$
return $(m_1 \neq m_2 \wedge h_1 = h_2)$

**Game** $\mathsf{CR}^{f}$ :
$(p_1, p_2) \leftarrow \mathcal{B}();$
return $\mathsf{is\_coll}(p_1, p_2)$

**Adversary** $\mathcal{B}()$ :
$(m_1, m_2) \leftarrow \mathcal{A}();$
[. . . *code omitted*. . . ]
return $(p_1, p_2)$

- ✓ prove logical judgments to establish equivalences between games

$$\vdash \mathsf{CR}^{\mathsf{MD}} \sim \mathsf{CR}^{f} : \mathsf{true} \Rightarrow \mathsf{res}\langle 1 \rangle \rightarrow \mathsf{res}\langle 2 \rangle$$

- ✓ derive inequalities between probabilities of events in games

$$\Pr\left[\mathsf{CR}^{\mathsf{MD}} : \mathsf{res}\right] \leq \Pr\left[\mathsf{CR}^{f} : \mathsf{res}\right]$$

# What is indifferentiability?

- Construction behaves like a RO (exception multi-stage properties)
  - Implies bounds on collision resistance, preimage resistance, second-preimage resistance, resistance to length-extension attacks

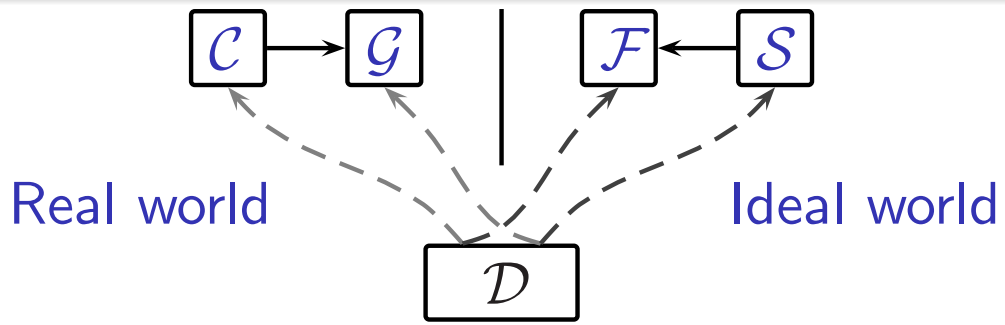
# What is indifferentiability?

- Construction behaves like a RO (exception multi-stage properties)
  - Implies bounds on collision resistance, preimage resistance, second-preimage resistance, resistance to length-extension attacks
- Distinguisher $\mathcal{D}$ runs in two possible settings

Real world: $\mathcal{C}$ Merkle-Damgård hash construction
$\mathcal{G}$ *ideal* compression function
Ideal world: $\mathcal{F}$ *ideal* hash function
$\mathcal{S}$ simulator we have to design

$\mathcal{C}^{\mathcal{G}}$ is indifferentiable from $\mathcal{F} \iff \exists\, \mathcal{S}$ such that $\forall\, \mathcal{D}$,

$$\left|\Pr[\mathcal{D}^{\mathcal{C},\mathcal{G}} = 1] - \Pr[\mathcal{D}^{\mathcal{F},\mathcal{S}} = 1]\right| < \epsilon$$



# What is indifferentiability?

**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

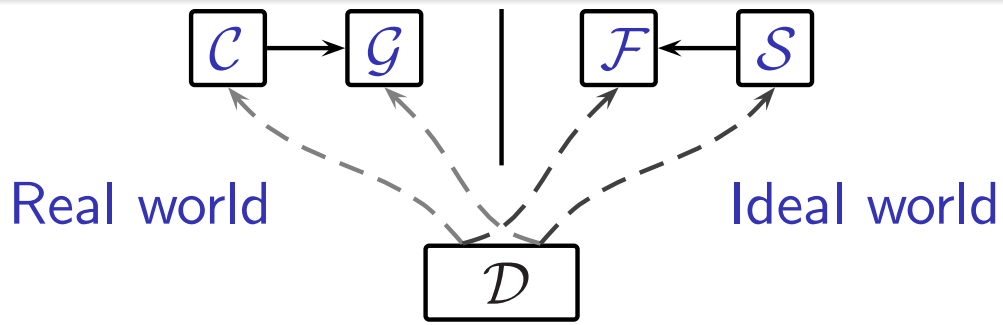
- Distinguisher $\mathcal{D}$ runs in two possible settings

Real world: $\mathcal{C}$ Merkle-Damgård hash construction
$\mathcal{G}$ *ideal* compression function

Ideal world: $\mathcal{F}$ *ideal* hash function
$\mathcal{S}$ simulator we have to design

$\mathcal{C}^{\mathcal{G}}$ is indifferentiable from $\mathcal{F} \iff \exists\, \mathcal{S}$ such that $\forall\, \mathcal{D}$,

$$\left|\Pr[\mathcal{D}^{\mathcal{C},\mathcal{G}} = 1] - \Pr[\mathcal{D}^{\mathcal{F},\mathcal{S}} = 1]\right| < \epsilon$$



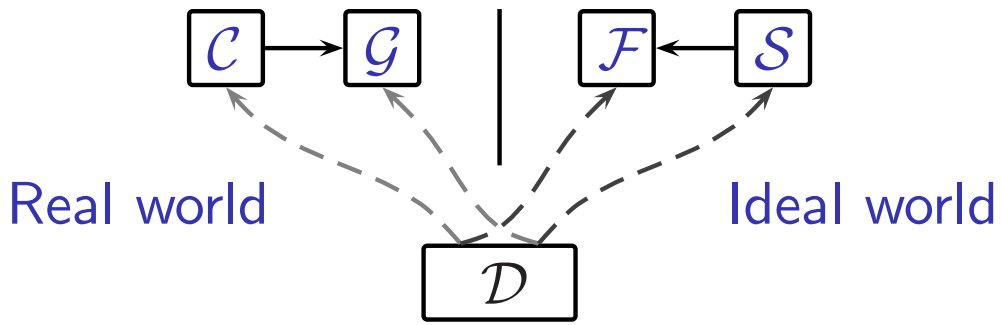
# What is indifferentiability?

**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.



# What is indifferentiability?

**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

# What is indifferentiability?

**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\text{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.

# What is indifferentiability?

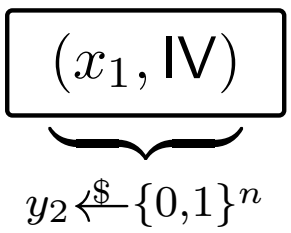
**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\text{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.



# What is indifferentiability?

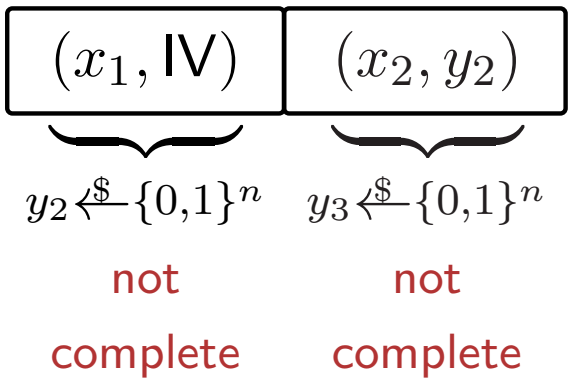
**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\text{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.



# What is indifferentiability?

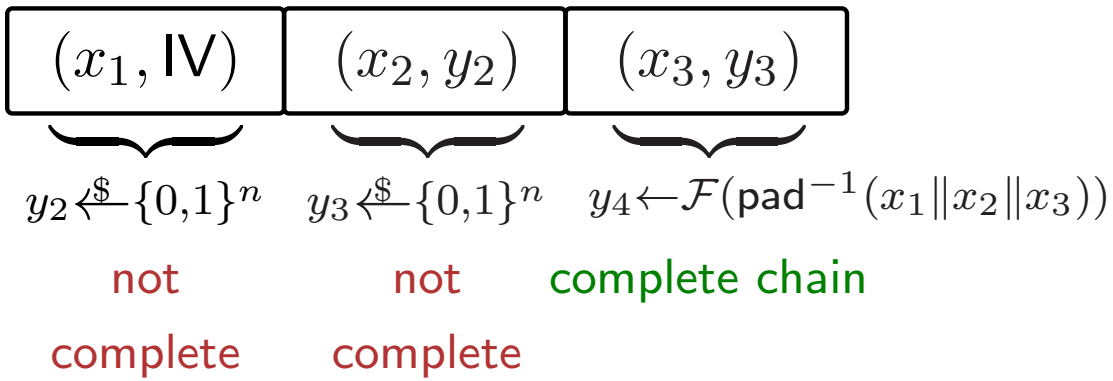
**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\mathsf{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.



# What is indifferentiability?

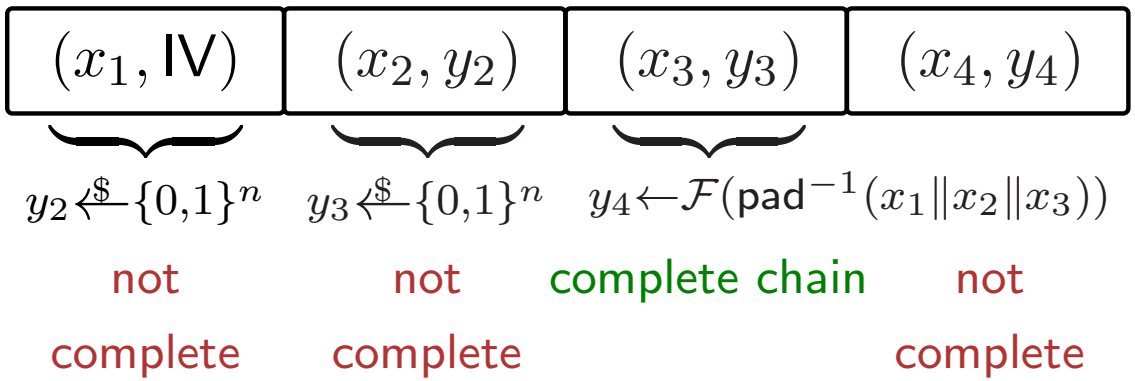
**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\mathsf{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.



# What is indifferentiability?

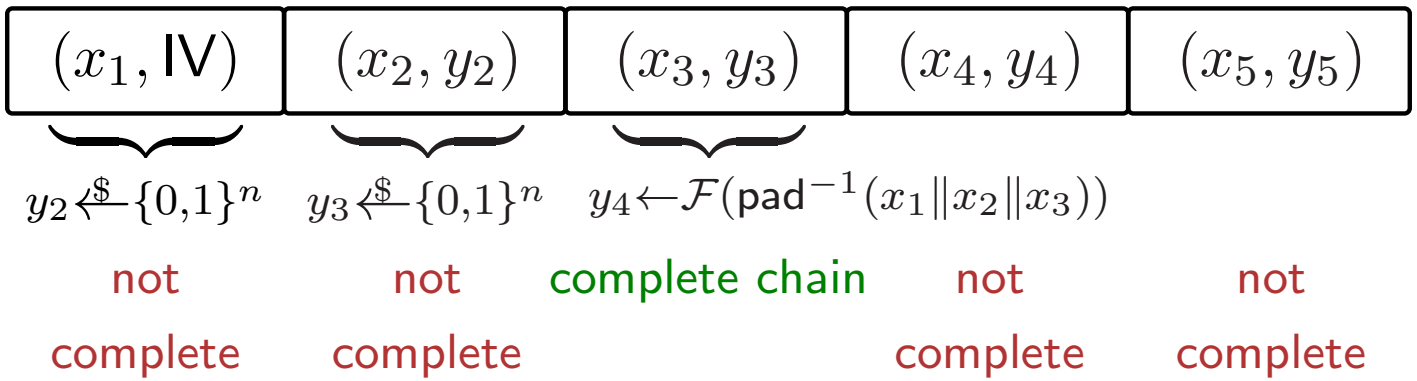
**Theorem [Coron et. al, 2005]**

Prefix-free Merkle-Damgård is indifferentiable from an ideal hash function.

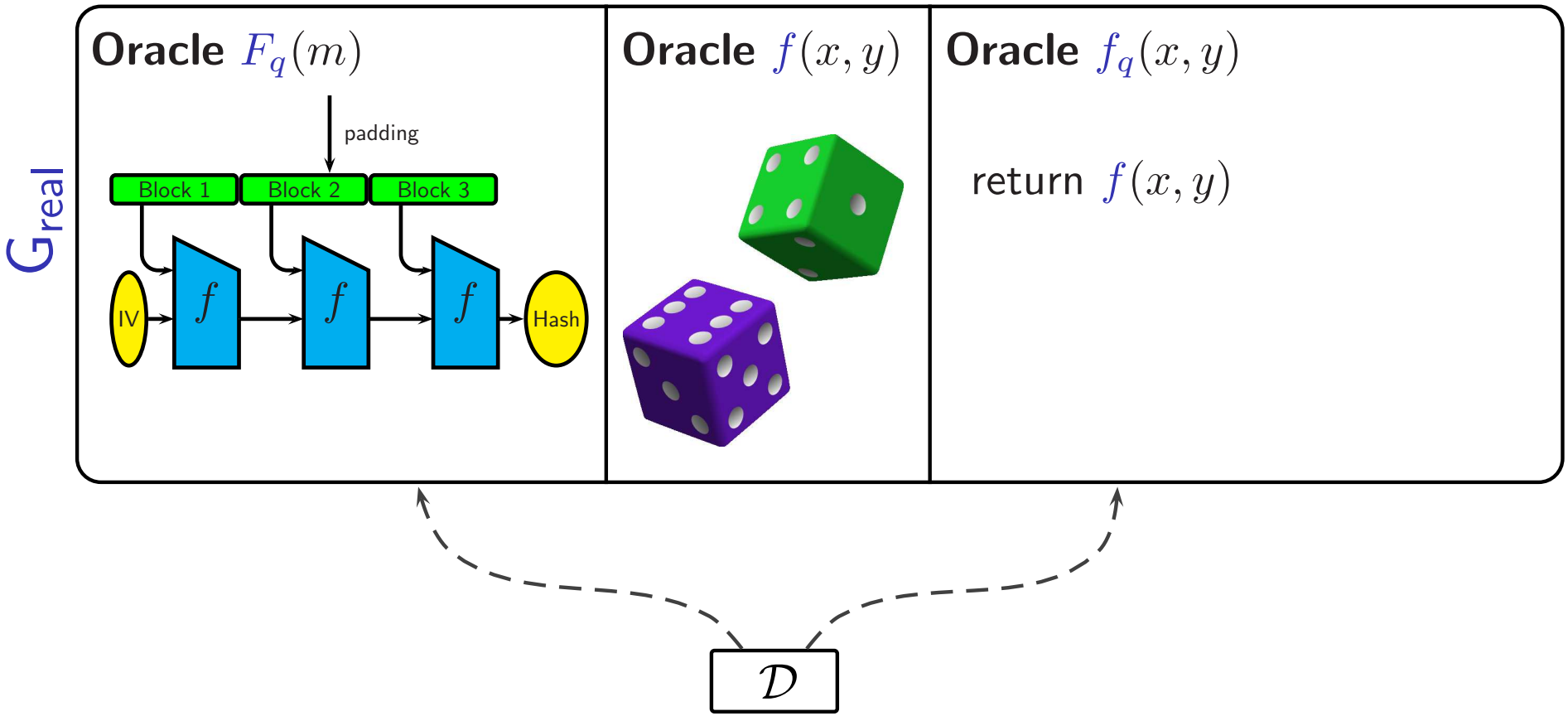
Proof idea for simulator $\mathcal{S}$:

- Keep track of queries to identify complete Merkle-Damgård chains
- For ends of chains query random oracle $\mathcal{F}$
- Otherwise answer queries randomly
- Prefix-freeness: $\mathcal{F}$ cannot be used to check intermediate chain values.

Let $\mathsf{pad}(m) = (x_1, x_2, x_3)$ be the padding of message $m$.



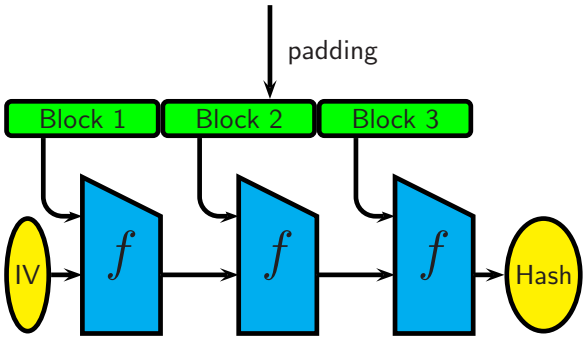
# Implementing the two worlds



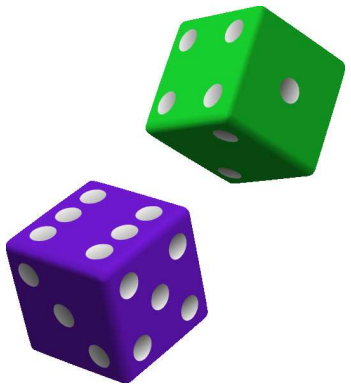
# Implementing the two worlds

$G_{\text{real}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



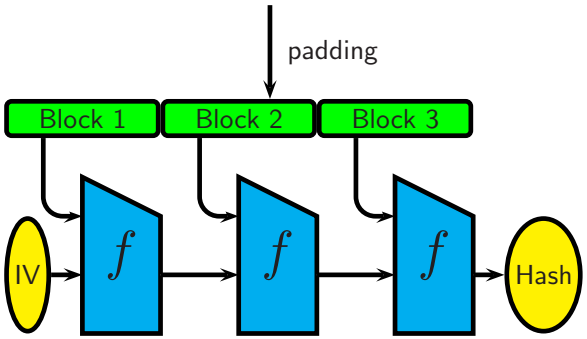
**Oracle** $f_q(x, y)$

return $f(x, y)$

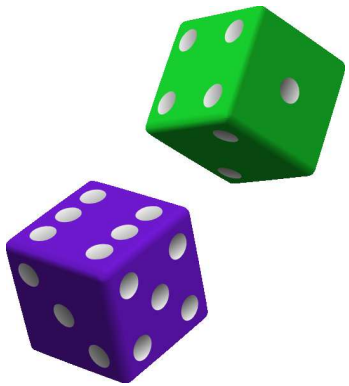
# Implementing the two worlds

$G_{real}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



**Oracle** $f_q(x, y)$

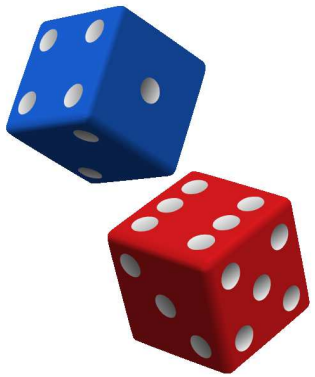
return $f(x, y)$

$G_{ideal}$

**Oracle** $F_q(m)$

return $RO(m)$

**Oracle** $RO(m)$



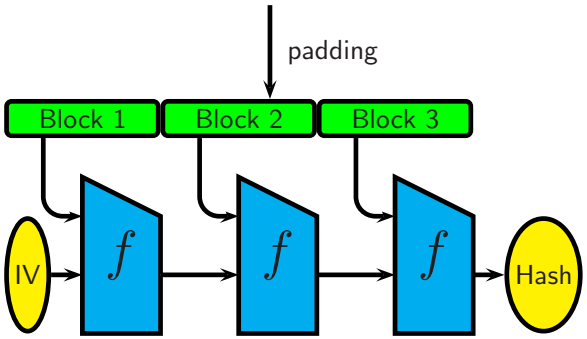
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow$ find_chain$(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \xleftarrow{\$} \{0, 1\}^n$;
return $\boldsymbol{T}[x, y]$

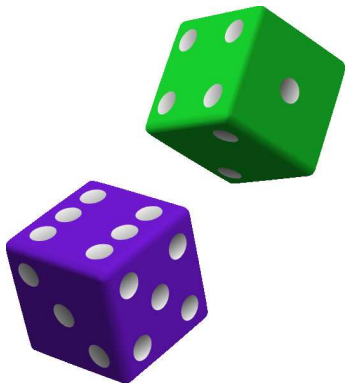
# Step 1: Check for bad events and introduce find_chain

$G_{\text{real}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



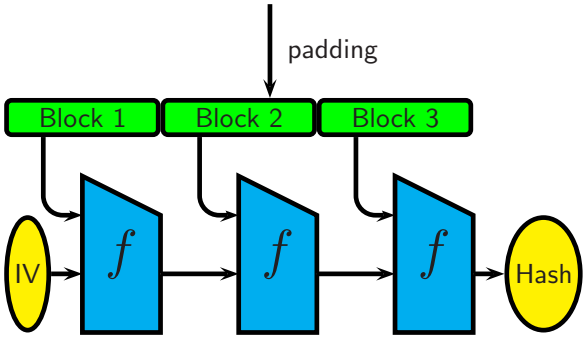
**Oracle** $f_q(x, y)$

return $f(x, y)$

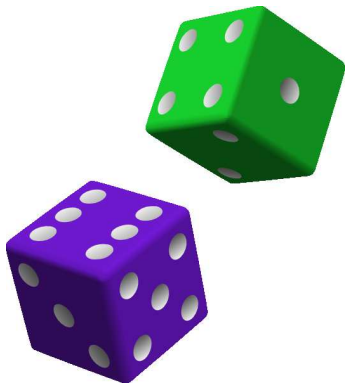
# Step 1: Check for bad events and introduce find_chain

$G_{real}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

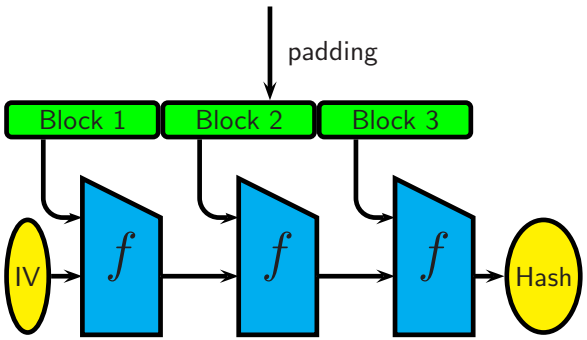


**Oracle** $f_q(x, y)$

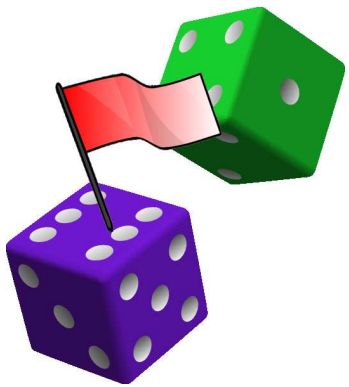
return $f(x, y)$

$G_{realbad}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



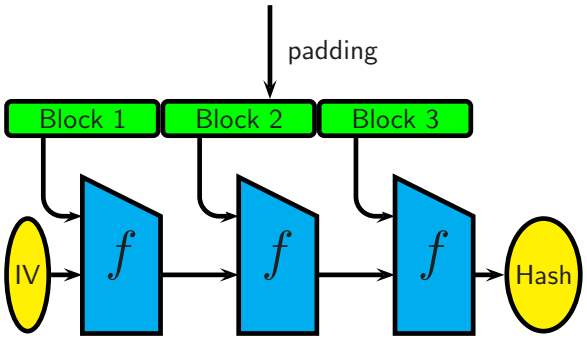
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \mathsf{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
return $\boldsymbol{T}[x, y]$

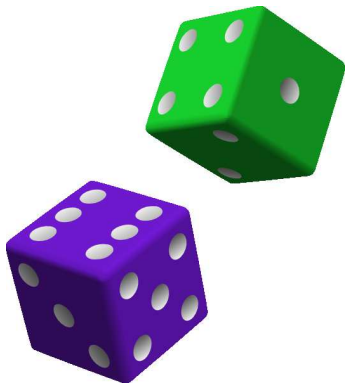
# Step 1: Check for bad events and introduce find_chain

$G_{real}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

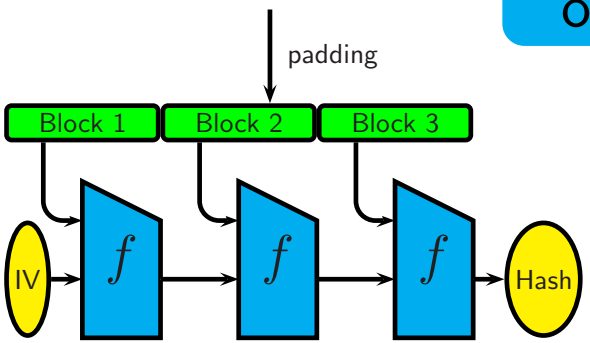


**Oracle** $f_q(x, y)$

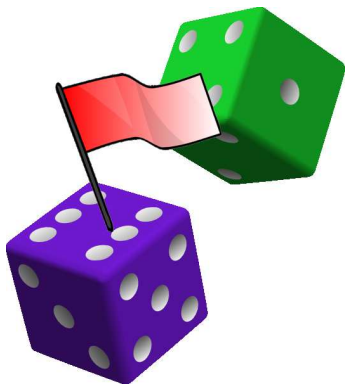
return $f(x, y)$

$G_{realbad}$

**Oracle** $F_q(m)$




No change in observable behavior

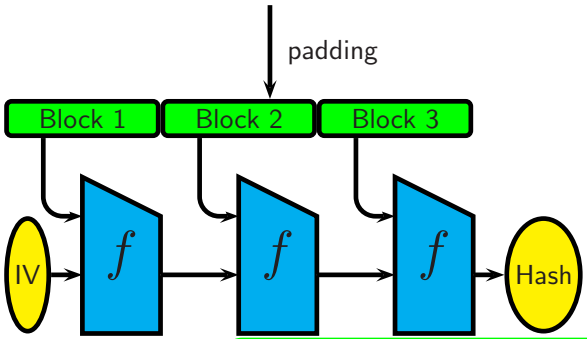


**acle** $f_q(x, y)$ :
$(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
$m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
if $m \neq$ none then
$\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
else
$\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
return $\boldsymbol{T}[x, y]$

# Step 1: Check for bad events and introduce find_chain

$\mathsf{G}_{\text{real}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

**Oracle** $f_q(x, y)$

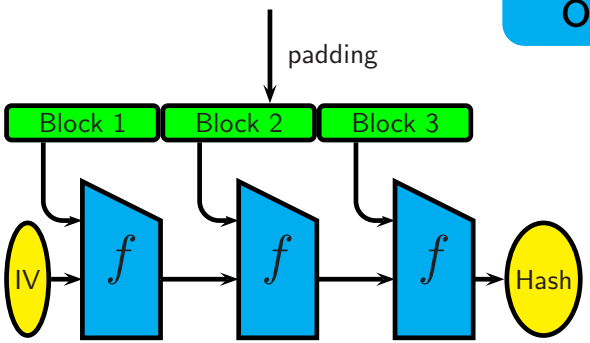
return $f(x, y)$

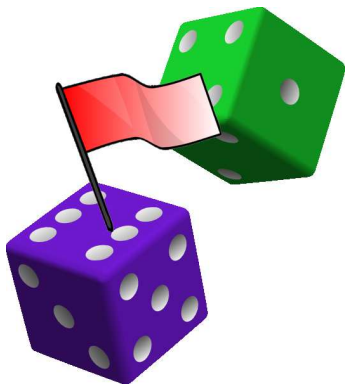
$$\Pr\left[\mathsf{G}_{\text{real}} : \mathcal{D}^{F_q, f_q} = 1\right] = \Pr\left[\mathsf{G}_{\text{realbad}} : \mathcal{D}^{F_q, f_q} = 1\right]$$

No change in observable behavior

$\mathsf{G}_{\text{realbad}}$

**Oracle** $F_q(m)$



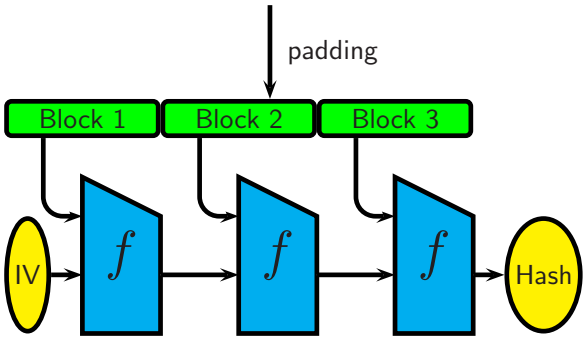



**acle** $f_q(x, y)$ :
$(x, y) \notin \mathsf{dom}(\boldsymbol{T})$ then
$m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
if $m \neq \mathsf{none}$ then
$\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
else
$\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
return $\boldsymbol{T}[x, y]$

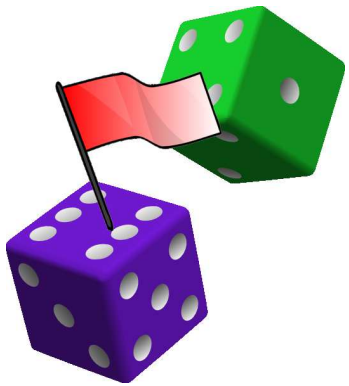
# Step 2: Call random oracle for complete chains

$G_{\text{realbad}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



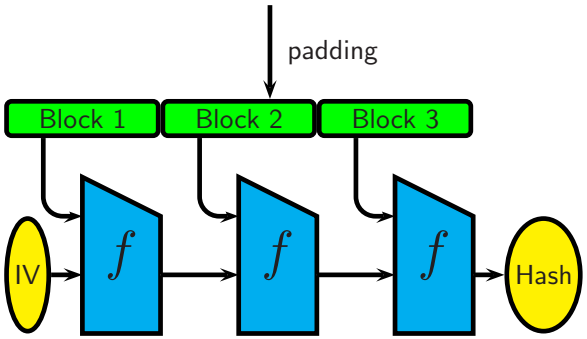
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \text{dom}(\boldsymbol{T})$ then
  $m \leftarrow \text{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \text{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
return $\boldsymbol{T}[x, y]$

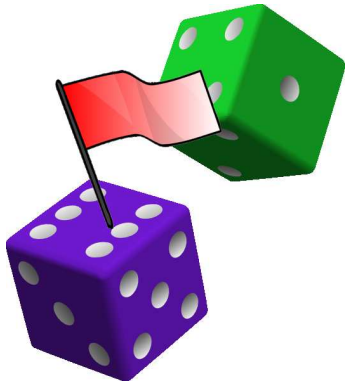
# Step 2: Call random oracle for complete chains

$G_{\text{realbad}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

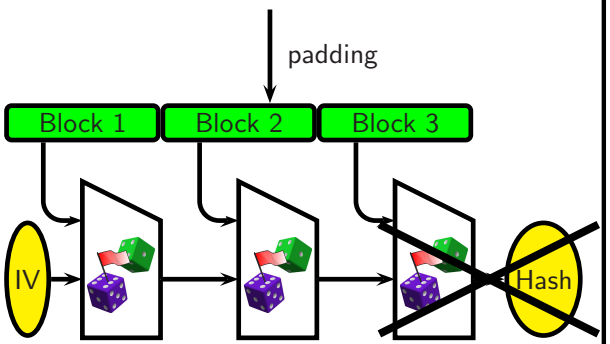


**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \text{dom}(\boldsymbol{T})$ then
  $m \leftarrow \text{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \text{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y)$;
return $\boldsymbol{T}[x, y]$

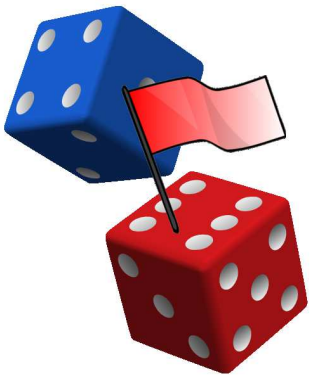
$G_{\text{realRO}}$

**Oracle** $F_q(m)$




return $RO(m)$

**Oracle** $RO(m)$



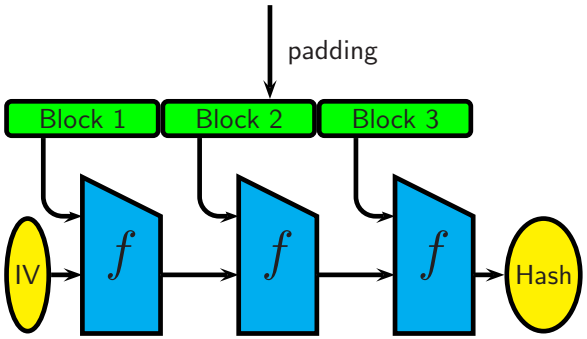
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \text{dom}(\boldsymbol{T})$ then
  $m \leftarrow \text{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \text{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y)$;
return $\boldsymbol{T}[x, y]$

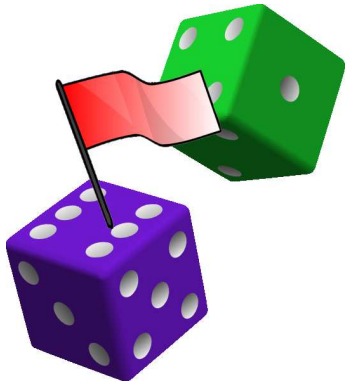
# Step 2: Call random oracle for complete chains

$G_{\text{realbad}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



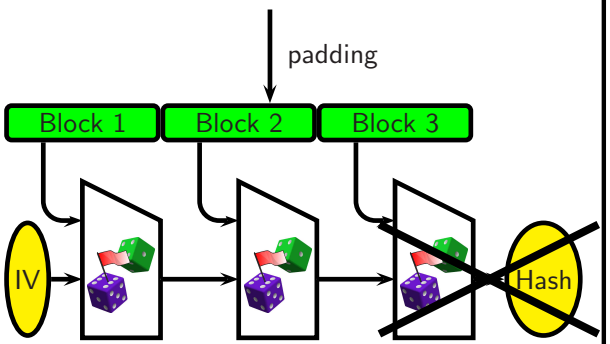
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y);$
return $\boldsymbol{T}[x, y]$

Prove games equal "upto bad"

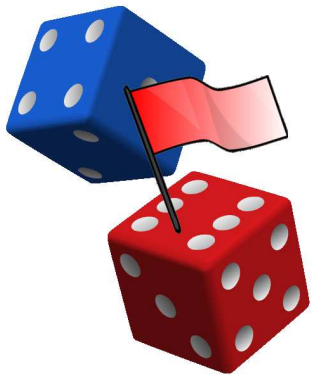
$G_{\text{realRO}}$

**Oracle** $F_q(m)$




return $RO(m)$

**Oracle** $RO(m)$



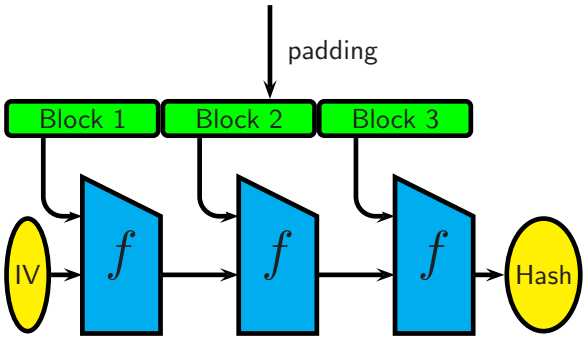
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m);$
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y);$
return $\boldsymbol{T}[x, y]$

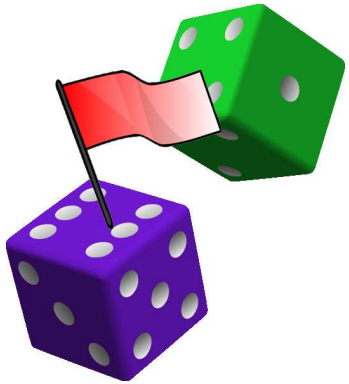
# Step 2: Call random oracle for complete chains

$G_{\text{realbad}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$



**Oracle** $f_q(x, y)$ :

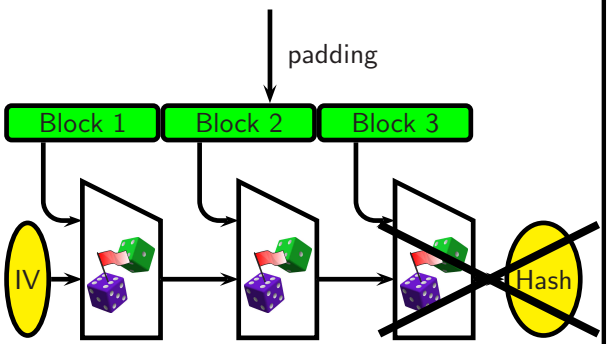
if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y);$
return $\boldsymbol{T}[x, y]$

Prove games equal "upto bad"

Complex loop invariants

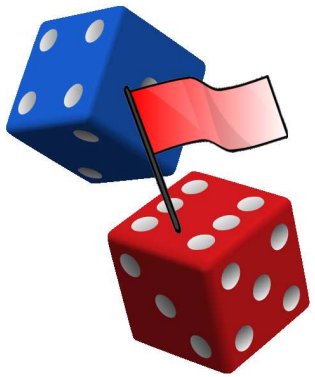
$G_{\text{realRO}}$

$n)$




return $RO(m)$

**Oracle** $RO(m)$



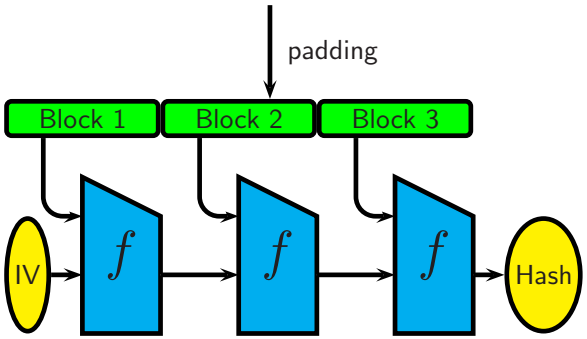
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m);$
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y);$
return $\boldsymbol{T}[x, y]$

# Step 2: Call random oracle for complete chains

$G_{\text{realbad}}$

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y);$
return $\boldsymbol{T}[x, y]$

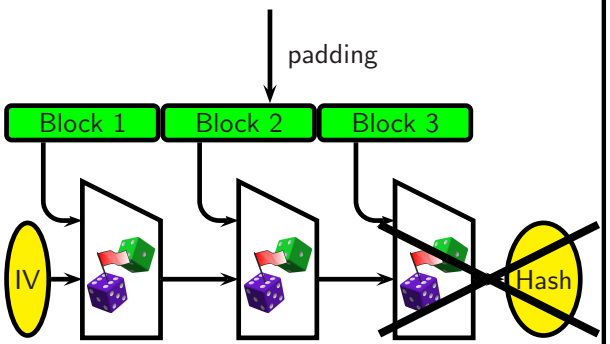
Support EasyCrypt with lemmas and axioms

Prove games equal "upto bad"

Complex loop invariants

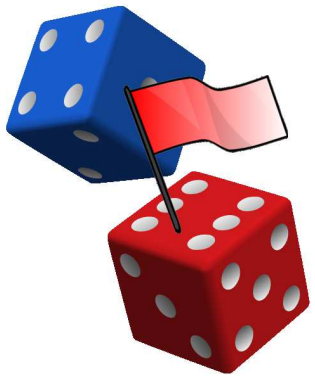
$G_{\text{realRO}}$

$n)$




return $RO(m)$

**Oracle** $RO(m)$

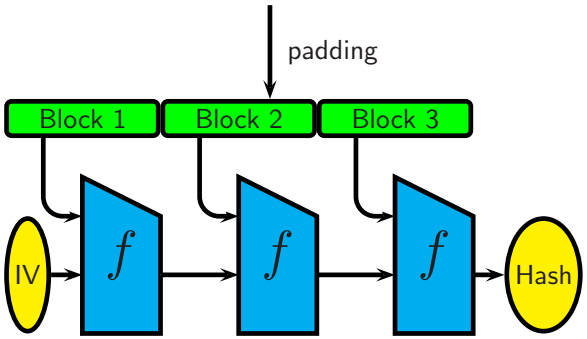


**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m);$
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y);$
return $\boldsymbol{T}[x, y]$

# Step 2: Call random oracle for complete chains

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y);$
return $\boldsymbol{T}[x, y]$

$G_{realbad}$

Support EasyCrypt with lemmas and axioms

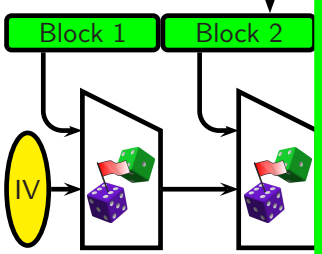
Prove games equal "upto bad"

Complex loop invariants

$G_{realRO}$

$n)$

pa



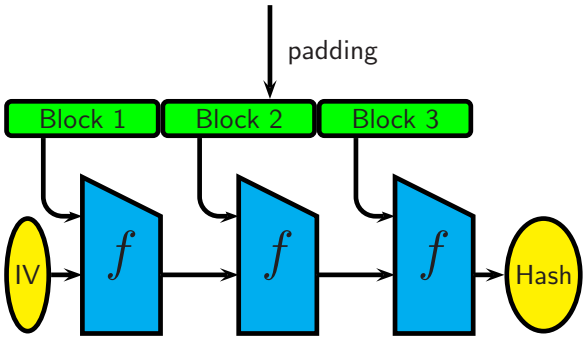

return $RO(m)$

return $\boldsymbol{T}[x, y]$

$$\left|\Pr\left[\mathsf{G}_{\mathsf{realbad}} : \mathcal{D}^{F_q, f_q} = 1\right] - \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \mathcal{D}^{F_q, f_q} = 1\right]\right| \leq \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \text{flag}\right]$$

# Step 2: Call random oracle for complete chains

**Oracle** $F_q(m)$




**Oracle** $f(x, y)$

**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x, y] \leftarrow f(x, y);$
return $\boldsymbol{T}[x, y]$

$G_{realbad}$

Support EasyCrypt with lemmas and axioms

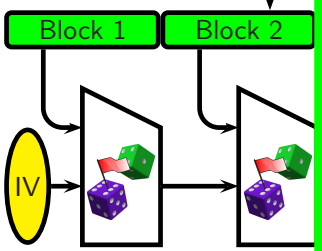
Prove games equal "upto bad"

Complex loop invariants

$G_{realRO}$

$m)$

pa




return $RO(m)$

return $\boldsymbol{T}[x, y]$

$$\left|\Pr\left[\mathsf{G}_{\mathsf{realbad}} : \mathcal{D}^{F_q, f_q} = 1\right] - \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \mathcal{D}^{F_q, f_q} = 1\right]\right|$$

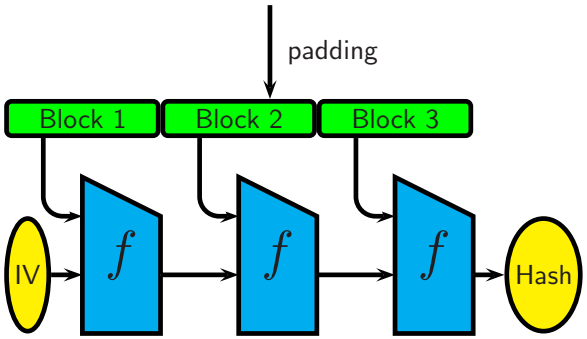
$$\leq \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \text{flag}\right]$$

Bound probabilities of bad events

# Step 2: Call random oracle for complete chains

$G_{realbad}$

**Oracle** $F_q(m)$




**Oracle** $f(x,y)$

**Oracle** $f_q(x,y)$ :

if $(x,y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T},x,y)$
  if $m \neq$
    $\boldsymbol{T}[x,$
  else
    $\boldsymbol{T}[x,y] \leftarrow f(x,y);$
return $\boldsymbol{T}[x,y]$

Support EasyCrypt with lemmas and axioms

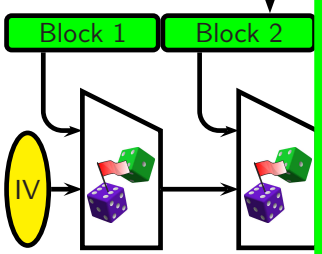
Prove games equal "upto bad"

Complex loop invariants

$G_{realRO}$

$m)$

pa




return $RO(m)$

return $\boldsymbol{T}[x,y]$

$$\left|\Pr\left[\mathsf{G}_{\mathsf{realbad}} : \mathcal{D}^{F_q,f_q} = 1\right] - \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \mathcal{D}^{F_q,f_q} = 1\right]\right|$$

$$\leq \Pr\left[\mathsf{G}_{\mathsf{realRO}} : \text{flag}\right]$$

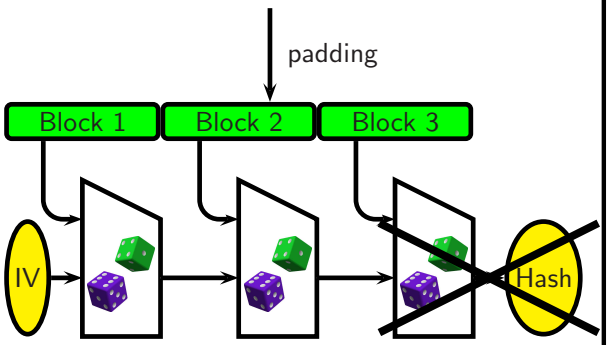
$$\leq \frac{3q^2}{2^n}$$

Bound probabilities of bad events

# Step 3: From eager to lazy sampling

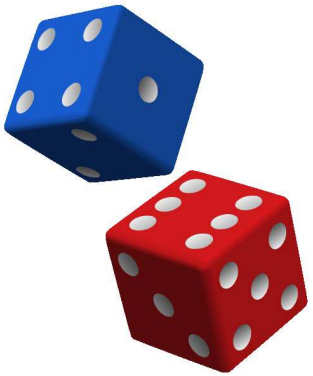
$G_{\text{realRO}}$

**Oracle** $F_q(m)$




return $RO(m)$

**Oracle** $RO(m)$



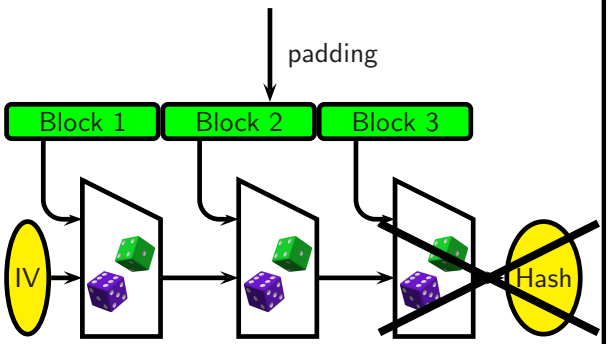
**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \text{dom}(\boldsymbol{T})$ then
  $m \leftarrow \text{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \text{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y)$;
return $\boldsymbol{T}[x, y]$

# Step 3: From eager to lazy sampling

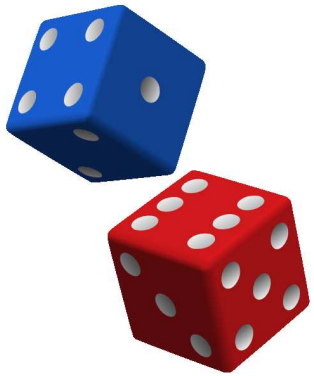
$G_{\text{realRO}}$

**Oracle** $F_q(m)$




return $RO(m)$

**Oracle** $RO(m)$



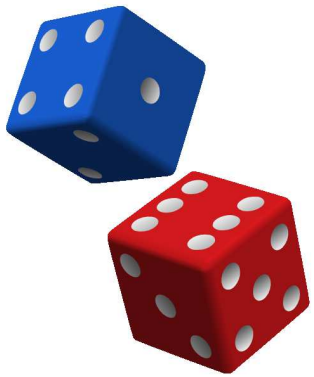
**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y)$;
return $\boldsymbol{T}[x, y]$

$G_{\text{ideal}}$

**Oracle** $F_q(m)$

return $RO(m)$

**Oracle** $RO(m)$

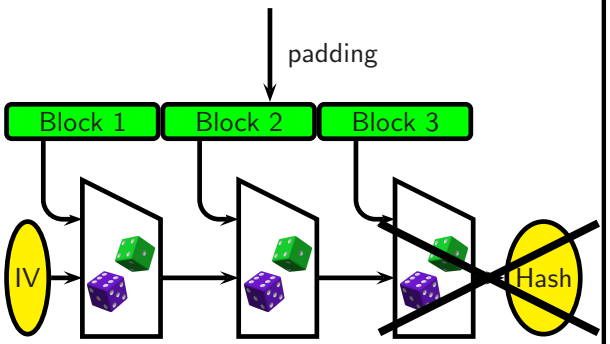


**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \xleftarrow{\$} \{0,1\}^n$;
return $\boldsymbol{T}[x, y]$

# Step 3: From eager to lazy sampling

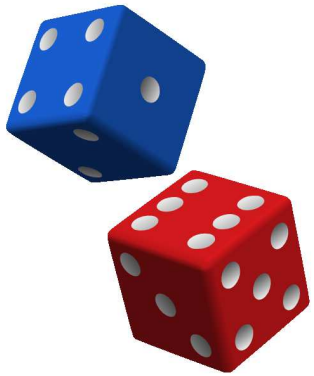
$G_{\text{realRO}}$

**Oracle** $F_q(m)$




return $RO(m)$

**Oracle** $RO(m)$



**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \mathrm{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \leftarrow$  $(x, y)$;
return $\boldsymbol{T}[x, y]$

$G_{\text{ideal}}$

**Oracle** $F_q(m)$

return $RO(m)$

**Oracle** $RO(m)$

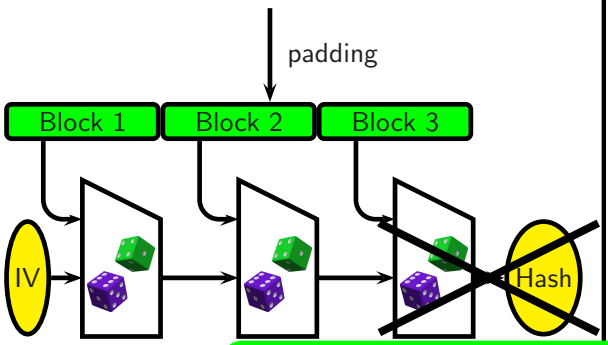
**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathrm{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathrm{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq \mathrm{none}$ then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \xleftarrow{\$} \{0,1\}^n$;
return $\boldsymbol{T}[x, y]$

Resampling values unknown to $\mathcal{D}$ does not change observable behavior

# Step 3: From eager to lazy sampling

$\mathsf{G}_{\mathsf{realRO}}$

**Oracle** $F_q(m)$




return $R$

**Oracle** $RO(m)$

**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathsf{dom}(\boldsymbol{T})$ then
$m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
if $m \neq \mathsf{none}$ then
$\boldsymbol{T}[x, y] \leftarrow RO(m)$;
else
;

$$\Pr\left[\mathsf{G}_{\mathsf{realRO}} : \mathcal{D}^{F_q, f_q} = 1\right] = \Pr\left[\mathsf{G}_{\mathsf{ideal}} : \mathcal{D}^{F_q, f_q} = 1\right]$$

$\mathsf{G}_{\mathsf{ideal}}$

**Oracle** $F_q(m)$

return $RO(m)$

**Oracle** $RO(m)$

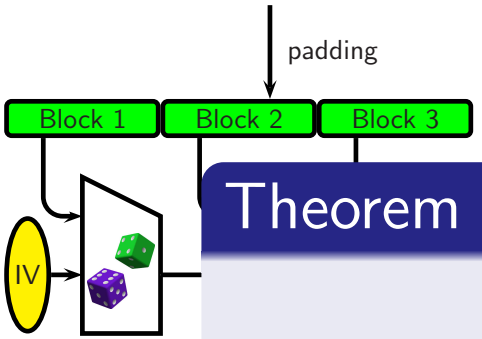
**Oracle** $f_q(x, y)$ :
if $(x, y) \notin \mathsf{dom}(\boldsymbol{T})$ then
$m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
if $m \neq \mathsf{none}$ then
$\boldsymbol{T}[x, y] \leftarrow RO(m)$;
else
$\boldsymbol{T}[x, y] \xleftarrow{\$} \{0,1\}^n$;
return $\boldsymbol{T}[x, y]$

Resampling values unknown to $\mathcal{D}$ does not change observable behavior

# Step 3: From eager to lazy sampling

$\mathsf{G}_{\mathsf{realRO}}$

**Oracle** $F_q(m)$




return $R$

**Oracle** $RO(m)$

**Oracle** $f_q(x, y)$ :

if $(x, y) \notin \mathsf{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then

**Theorem (Indifferentiability)**

$$\left|\Pr\left[\mathsf{G}_{\mathsf{real}} : \mathcal{D}^{F_q, f_q} = 1\right] - \Pr\left[\mathsf{G}_{\mathsf{ideal}} : \mathcal{D}^{F_q, f_q} = 1\right]\right| \leq \frac{3q^2}{2^n}$$

$\mathsf{G}_{\mathsf{ideal}}$

**Oracle** $F$

return $RO(m)$

if $(x, y) \notin \mathsf{dom}(\boldsymbol{T})$ then
  $m \leftarrow \mathsf{find\_chain}(\boldsymbol{T}, x, y)$
  if $m \neq$ none then
    $\boldsymbol{T}[x, y] \leftarrow RO(m)$;
  else
    $\boldsymbol{T}[x, y] \xleftarrow{\$} \{0,1\}^n$;
return $\boldsymbol{T}[x, y]$

Resampling values unknown to $\mathcal{D}$ does not change observable behavior

# Conclusion & Future work

We have formally verified

- Collision resistance of suffix-free MD
  - 150 lines in EasyCrypt
- Indifferentiability of prefix-free MD
  - 3600 lines in EasyCrypt + 3000 lines in Coq

Most results do not apply to SHA-3 finalists directly

- Final transformation, non-ideal compression function, not prefix-free

But: Interesting non-trivial result

- Excellent starting point for formal security proofs the SHA-3 finalists
- Proofs based on weaker assumptions not significantly different

# Thank you for your attention!

Questions?