TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Security and Cryptography 1

Stefan Köpsell, Thorsten Strufe

Module 6: Integrity

Disclaimer: large parts from Mark Manulis, Dan Boneh, Stefan Katzenbeisser

Dresden, WS 17/18

Reprise from the last modules

You have an overview of cryptography and cryptology

You know different adversary models and their corresponding games

You know what symmetric cryptography is

You recall the difference of stream and block ciphers

You can explain the OTP and constructions for stream ciphers

You can prove that the OTP has perfect secrecy

You can tell PRFs and PRP apart and you know constructions for block ciphers

You can explain different modes of operation and their properties

Module Outline

Verification of message integrity as a goal

Adversary and security models

Hashes and cryptographic hash functions

Collisions and how to create them (also: the birthday paradox)

The Merkle-Damgard construction and some real hash functions (MD5, SHA-1)

Block ciphers as compression functions

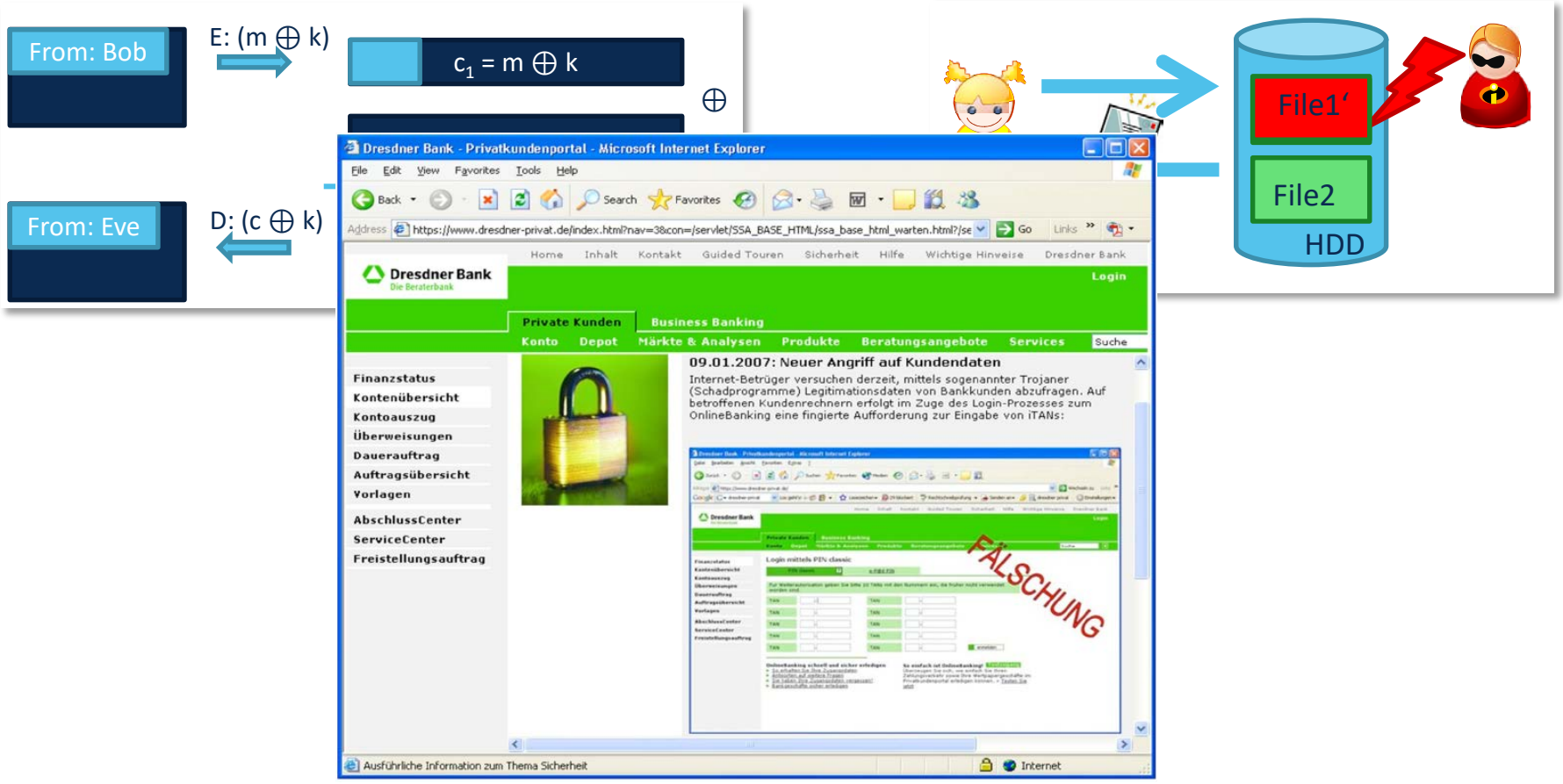
Secure MACs from hash functions and PRFs

MACs using block ciphers (CBC, NMAC, HMAC)

Integrity and Authenticity

So far messages can be kept confidential

Integrity of messages not given



Message Integrity

Algorithms:

Tag S: $M \rightarrow T$ $M = \{0,1\}^n$; $S = \{0,1\}^t$ with $n >> t$

Verify V: $M \times T \rightarrow \{yes,no\}$

Bad Examples

Cross sum:

f(x) : calculate the cross sum of all bytes in the message

Is this secure? Why (not)?

f(5 23) = f(23 5)

CRC:

tag ← CRC(m) ; Verify tag: return CRC(m) == tag

Is this secure? Why (not)?

Integrity requires a secret

Adversary can create new message and recompute CRC

Simple Encryption:

tag ← Enc(k,m) | $_{1,\ldots,6}$ Verify tag: return tag == Enc(k,m) | $_{1,\ldots,6}$

Is this secure? Why (not)?

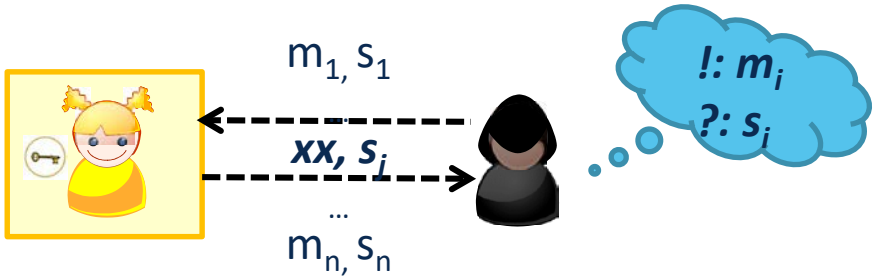
Security depends on |S|

Adversary can guess tag for a message in 2^6

Existential Forgery

Chosen Message Attack:

- given s_1, s_2, … s_n for chosen m_i



(variations are: known key / known signature attacks)

Existential Forgery:

Produce ***some*** new valid tuple (m,s) (any message, even gibberish)

⇒ adversary cannot produce a valid tag for a new message

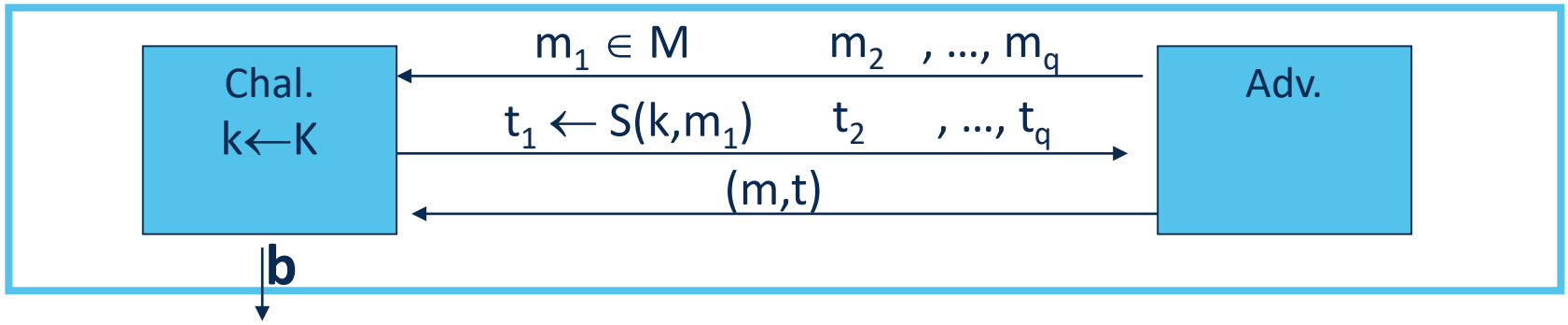
⇒ adversary cannot even produce (m,t') for (m,t) and t' ≠ t

Breaches in general:

Exist. forgery < selective forgery < universal forgery < total break

Defining the MAC game

For a MAC I=(S,V) and adversary A, where (S,V) additionally take *k:*



b=1 if V(k,m,t) = `yes' and $(m,t) \notin \{ (m_1,t_1) , \dots , (m_q,t_q) \}$

b=0 otherwise

Def: I=(S,V) is a **secure MAC** if for all "efficient" A:

$Adv_{MAC}[A,I] = Pr[\text{Chal. outputs 1}] \leq \varepsilon$

So how can we build a secure MAC?

Variations:

Existential forgery may forge tag on a message that seems gibberish

(like, for instance, a random-looking bitstring like a ciphertext or a key…)

Secure MAC from a PRF

Assume a PRF $F: K \times X \rightarrow Y$

Define integrity scheme $I_F = (S,V)$:

- $S(k,m) := F(k,m)$
- $V(k,m,t) :=$ test $t==F(k,m)$

- *Is this a secure integrity scheme?*

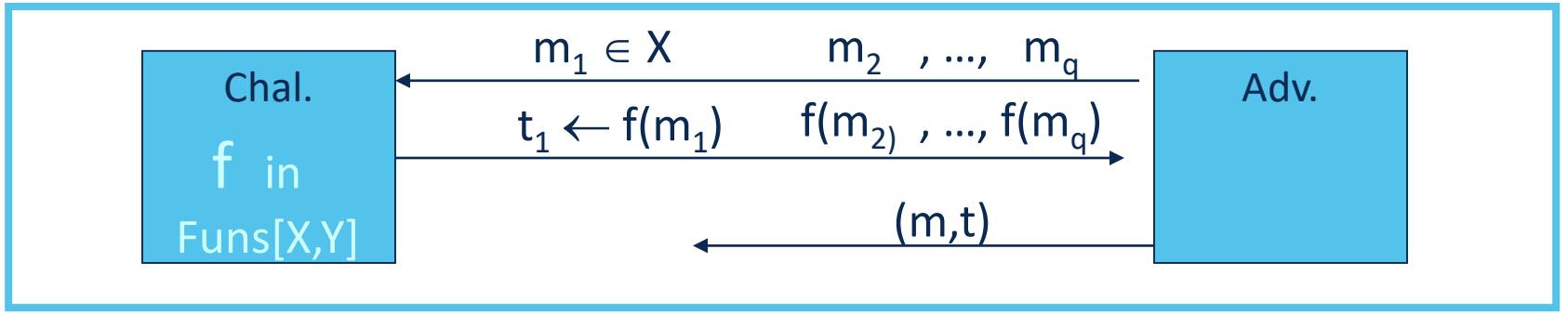
suppose F is PRF, m given, what is the chance of the adv. to guess s?

$$Adv_{MAC}[A,I_F] =$$

- *What are its downsides?*

Security of the simple PRF MAC

Assume **f: X⟶Y** is a random function and 1/|Y| is negligible



Assume **F: K x X⟶Y** is a secure PRF, construct MAC I_F:

For every efficient MAC adversary A attacking I_F, there must be an efficient PRF adversary B attacking F, s.t.:

$$Adv_{MAC}[A, I_F] \leq Adv_{PRF}[B, F] + 1/|Y|$$

What happens if we truncate the output?

$\Rightarrow I_F$ is secure as long as:

- F is a secure PRF (given), and |Y| is large ($|Y| \geq 2^{160}$)

Interlude: Collision Resistant Hash Functions

Goal:

Map a message of arbitrary length to a characteristic digest (fingerprint)

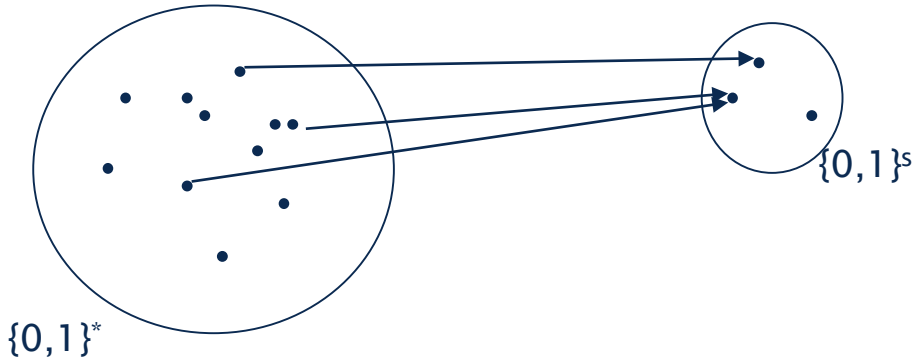
Hash H: M → S with $M = \{0,1\}^*$ and $S = \{0,1\}^s$

- has an efficient algorithm to evaluate H(x)
- is an „onto“ function (surjective, Im(H) = S)
- avoids collision (→ maps uniformly to S)
- creates chaos (slight changes in m yield large differences in s)

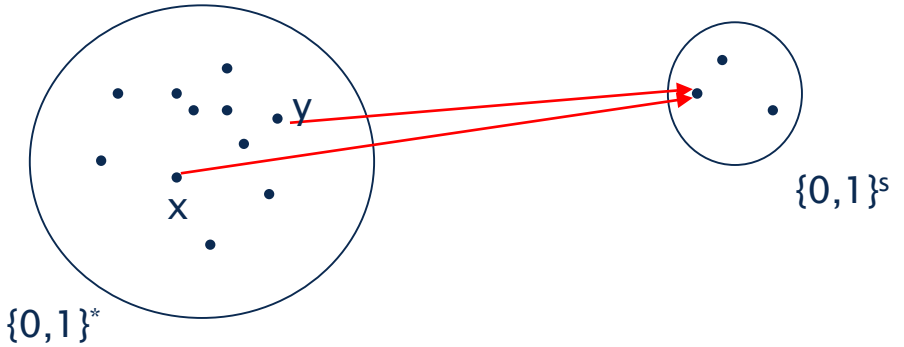
Cryptographic Hash Functions

Further properties / requirements of hash functions for security:

- Compression is irreversible



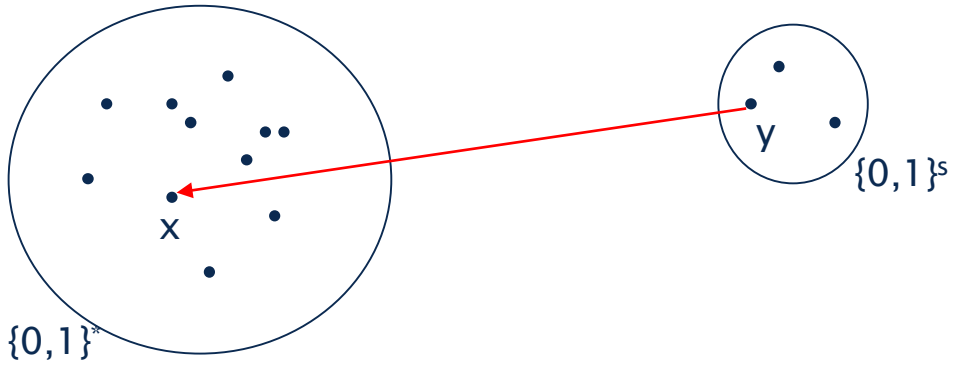
- Collision resistance



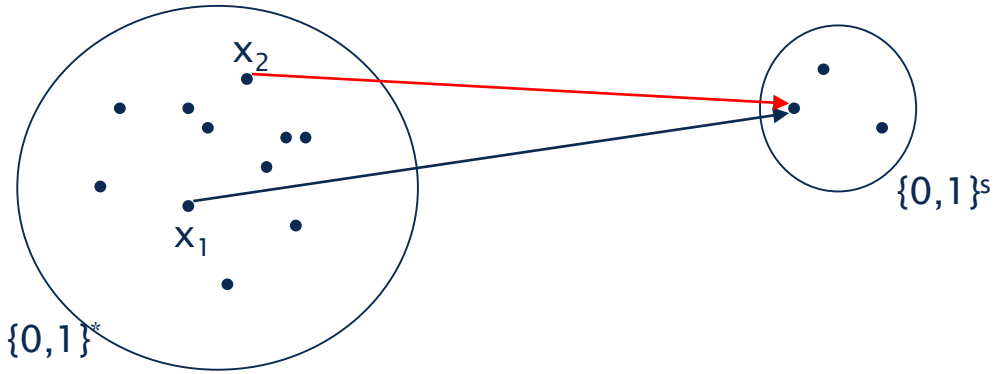
Cryptographic Hash Functions

Further requirements to hash functions for security:

- Pre-image resistance



- 2nd pre-image resistance



Collision Resistance

Let $H: M \rightarrow S$ be a hash function ($|M| >> |S|$)

A **collision** for H is a pair $m_0, m_1 \in M$ such that:

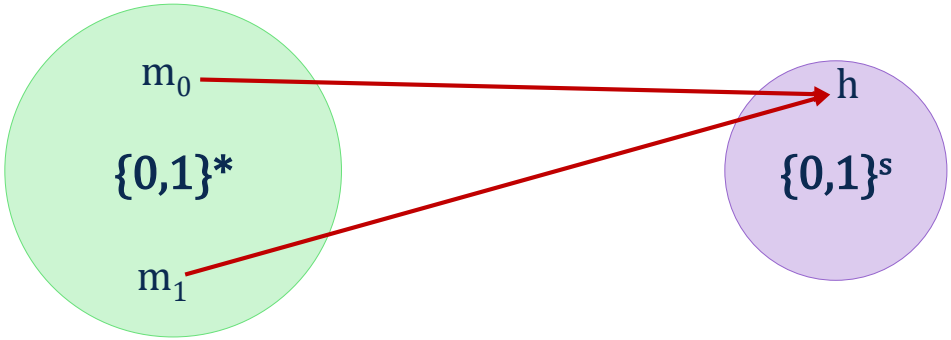
$$H(m_0) = H(m_1) \quad \text{and} \quad m_0 \neq m_1$$

A function H is **collision resistant** if for all (explicit) "eff" algs. A:

$$\mathbf{Adv_{CR}[A,H] = Pr[\ A\ outputs\ collision\ for\ H] \leq \varepsilon}$$

Brute Forcing Collisions

Do collisions exist?



Yes.

but it should be hard to find collisions (in polynomial time)

Trivial Collision-Finder (Brute Force)

s must be sufficiently large

compute $H \stackrel{\text{def}}{=} \{H(m) \mid \text{for all } m \in \{0,1\}^s\}$

if no collision found compute $H(m^*)$ for any $m^* \notin \{0,1\}^s$

time needed $O(2^s)$

there must be at least one m with $H(m) \in H$ such that $H(m) = H(m^*)$

Generic Attack on C.R. Hash Function

Let $H: M \rightarrow \{0,1\}^s$ be a hash function ($|M| >> 2^s$)

Generic alg. to find a collision **in time** $\mathbf{O(2^{s/2})}$ hashes

Algorithm:

1. Choose $\mathbf{2^{s/2}}$ random messages in M: $m_1, \ldots, m_{(2^{s/2})}$ (distinct w.h.p)
2. For i = 1, …, $2^{s/2}$ compute $t_i = H(m_i) \in \{0,1\}^s$
3. Look for a collision ($t_i = t_j$). If not found, got back to step 1.

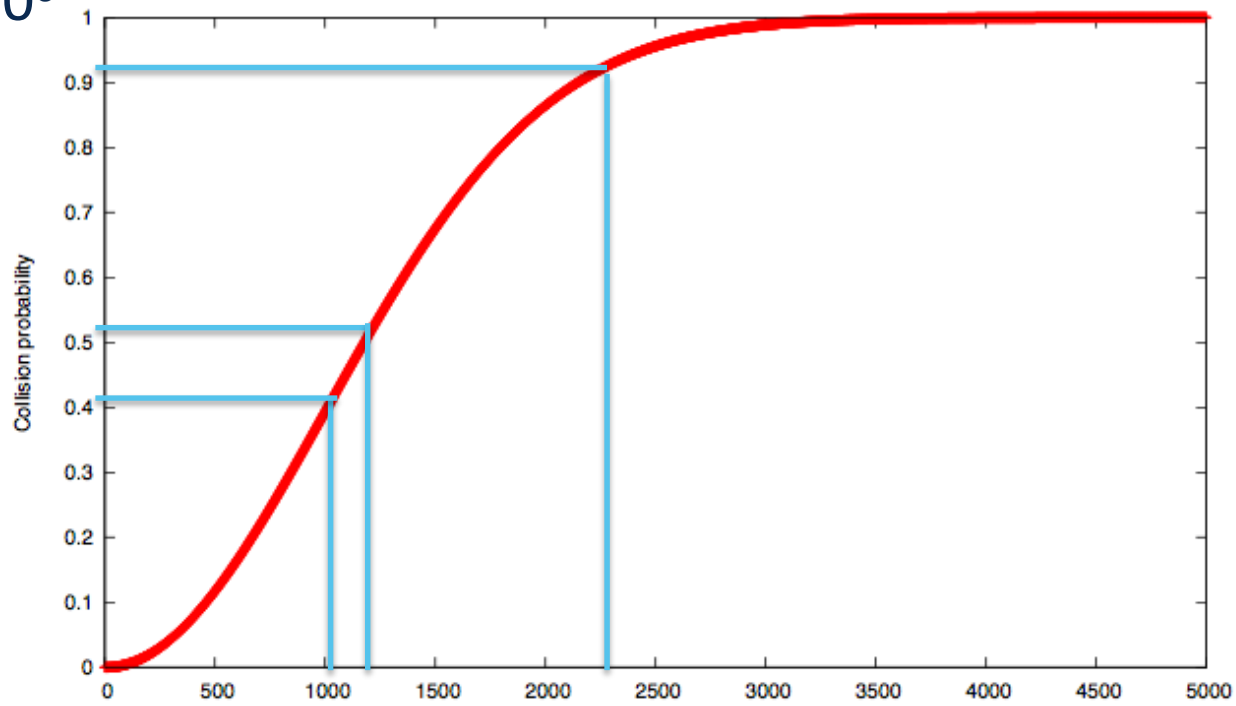
How well will this work?

The Birthday Paradox

Let $r_1, \ldots, r_n \in \{1,\ldots,B\}$ be random integers, chosen iid.

BP states: when $\mathbf{n} = 1.2 \times \mathbf{B^{1/2}}$ then $\Pr[\exists i \neq j: r_i = r_j] \geq ½$

$B=10^6$



Generic attack due to the B. P.

H: $M \rightarrow \{0,1\}^s$. Collision finding algorithm:

1. Choose $\mathbf{2^{s/2}}$ random elements in M: $m_1, \ldots, m_{2^{s/2}}$
2. For i = 1, …, $2^{s/2}$ compute $t_i = H(m_i) \in \{0,1\}^s$
3. Look for a collision $(t_i = t_j)$. If not found, got back to step 1.

Expected number of iterations until success ≈ 

Running time: $\mathbf{O(2^{n/2})}$ (space $O(2^{n/2})$)

Revisiting Requirements for Cryptographic h

Cryptographic hash functions:

Hash h: M → S with M = $\{0,1\}^*$ and S = $\{0,1\}^s$

Three core security requirements:

- **Collision** resistance
  - Collision: Given $h(\cdot)$, find m_1, m_2 with $m_1 \neq m_2$ such that $h(m_1) = h(m_2)$
  - $Adv_{CR}(A, h(\cdot))$: $Pr[Collision] \leq min\{|M|, |S|\}^{1/2}$
    - Common assumption: |M| >> |S| , hence $O(2^{s/2})$ (General attack due to BP)

- **Pre-image**, **second pre-image** resistance
  - Pre-image: given $h(m)$ find $h^{-1}(h(m))$ = m
  - 2nd pre-image: given m_1 find m_2 such that $m_1 \neq m_2$ and $h(m_1) = h(m_2)$
  - $Adv_{(S)PR}(A, h(\cdot)) \leq |S| + \varepsilon$, (cf. assumption above, hence $O(2^s)$)

The Merkle-Damgard construction

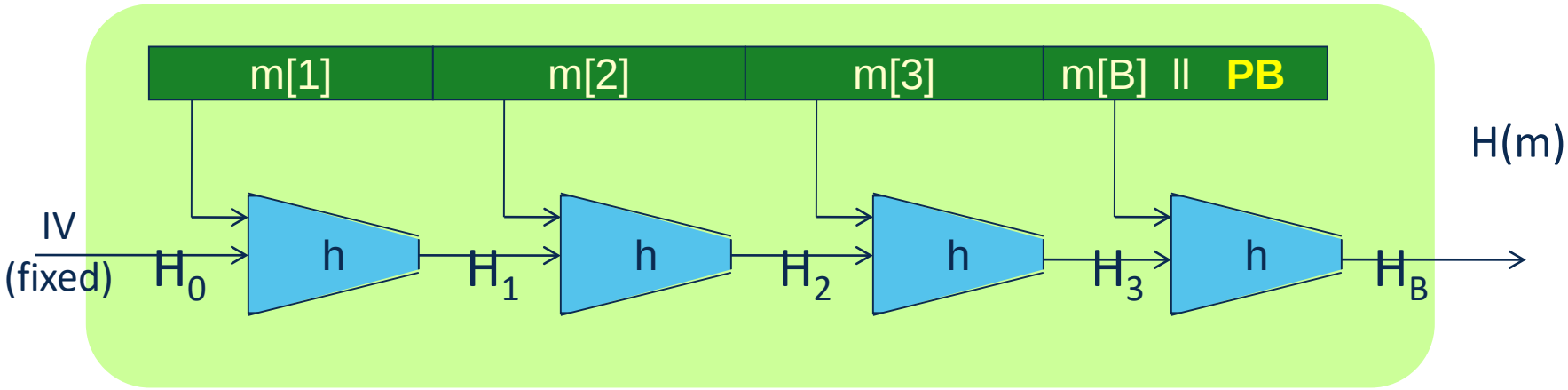
From *short message blocks* to *arbitrarily long messages*…

Given a compression function $h : \{0,1\}^{2s} \longrightarrow \{0,1\}^{s}$ and

Input $m \in \{0,1\}^*$ of length L and PB: = 1000…0 || L

64 bits

Construct H of B= $\lceil L/s \rceil$ iterations of h:



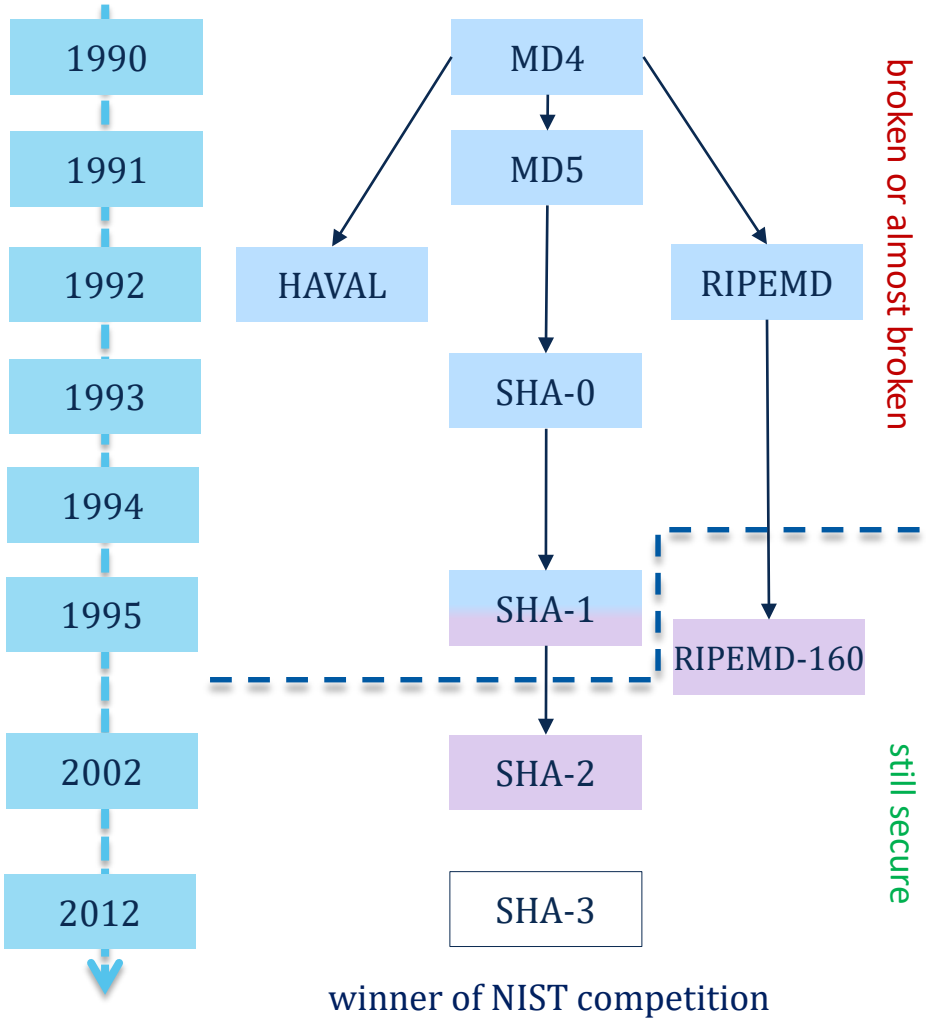
If h is a fixed length CRHF, then H is an arbitrary length CRHF

Proof: either M=M‘, or $H_{B-i}(m[B-i])=H_{B-i}(m'[B-i])$

no collision

collision on h

A brief history of Hash Functions



MD4 s = 128 bits
collisions in $O(2^8)$, preimages in $O(2^{102})$

MD5 s = 128 bits
collisions in $O(2^{32})$
known colliding documents, certificates

HAVAL s = 128, 160, 192, 224, 256 bits
collisions on HAVAL-128 in $O(2^6)$

RIPEMD s = 128 bits
collisions are known

SHA-0 s = 160 bits
collisions in $O(2^{39})$, replaced by SHA-1 in '95
meanwhile collisions in 1 hour

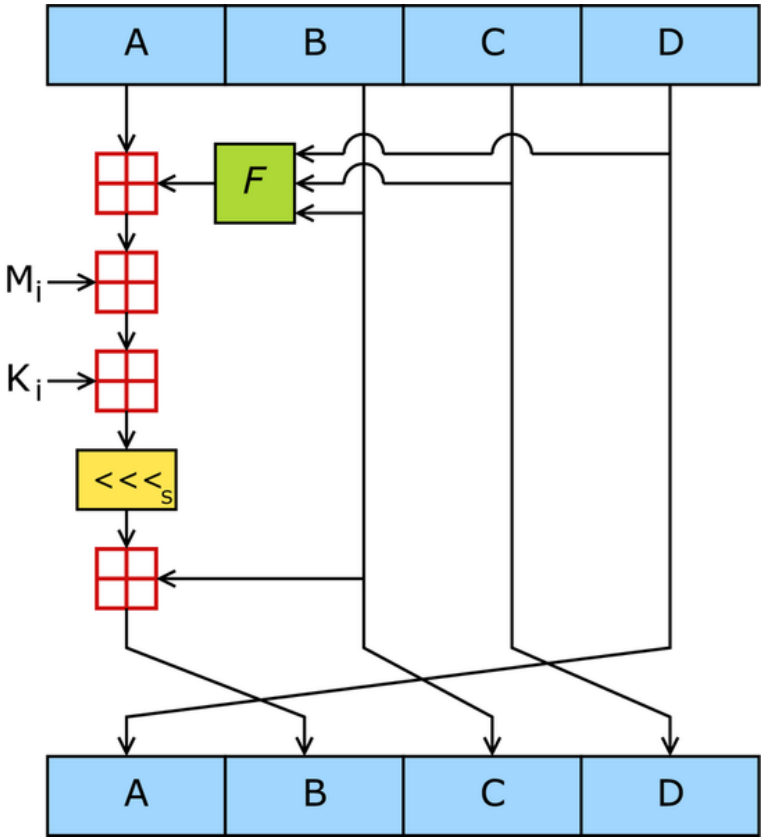
SHA-1 s = 160 bits
collisions in $O(2^{63})$ – $O(2^{69})$
still secure in practice

SHA-2 supports s = 224, 256, 384, 512 bits

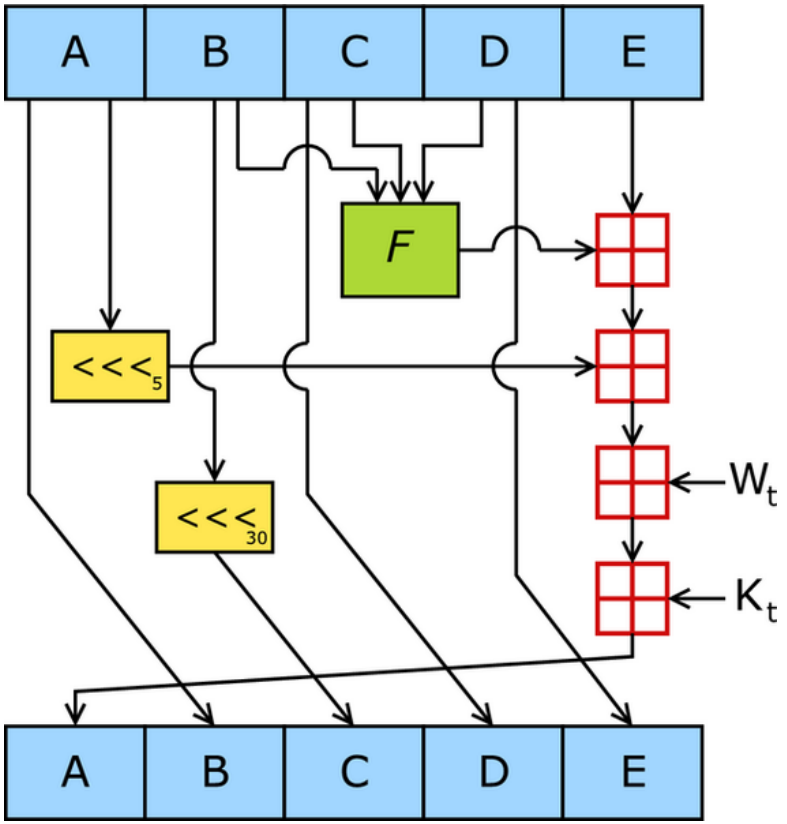
Popular Merkle Damgard constructions

While SHA-2 is also a MD construction, SHA-3 isn't...

Fixed length h (for arbitrary length m, as MD):



MD5 (128 bit)



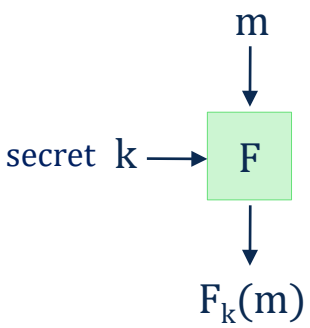
SHA-1 (160 bit)

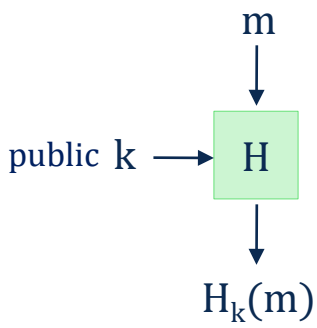
Hash Functions from Block Ciphers

Can we build compression functions from block ciphers?

- block ciphers $F : \{0,1\}^k \times \{0,1\}^b \longrightarrow \{0,1\}^b$ behave like PRPs
- the outputs of F are close to uniform
- collision-resistance
- provable security: reduction to the security of a block cipher
- block ciphers used in practice are efficient
- implementation of a block cipher immediate hash function

but what about the keys?



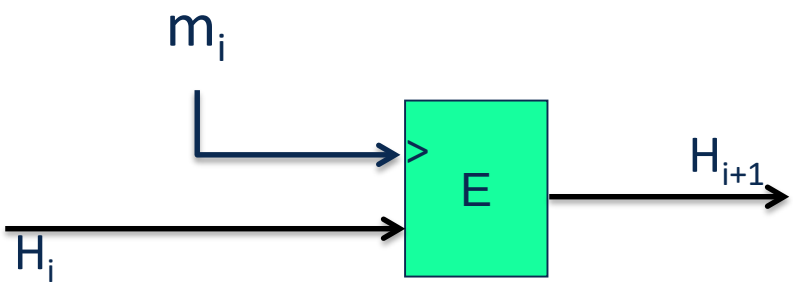


However, block ciphers are reversible!

An insecure attempt…

E: K× {0,1}^{n} ⟶ {0,1}^{n} a block cipher.

Construct cascade, for compression encrypt message blocks:



What's wrong with that?

H_{i+1}= E(m_i,H_i)

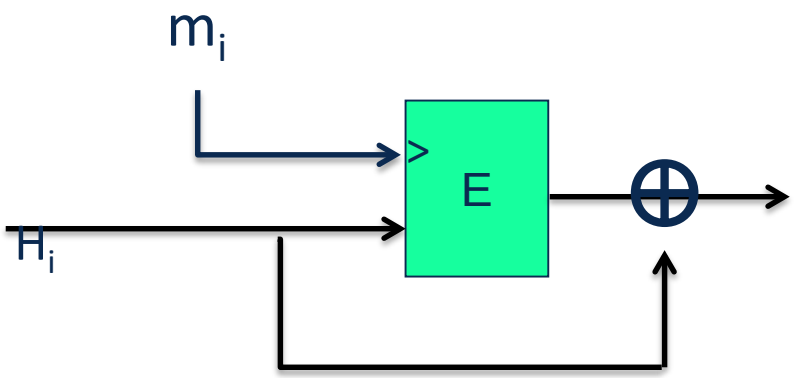
Can you find a collision on this compression function?

H_{i+1}= E(m', D(m',H_{i+1}))

Davies – Meyer Compression Function

E: K× $\{0,1\}^n \longrightarrow \{0,1\}^n$ a block cipher.

The **Davies-Meyer** compression function: **$h(H, m) = E(m, H) \oplus H$**



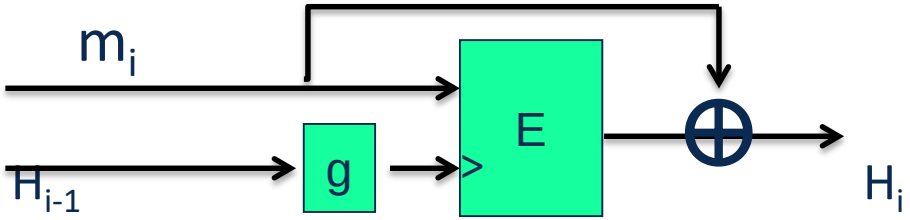
Thm: Suppose E is ideal cipher (collection of |K| random perms.).

To find collision **h(H,m)=h(H',m')** takes **$O(2^{n/2})$** evaluations of (E,D).

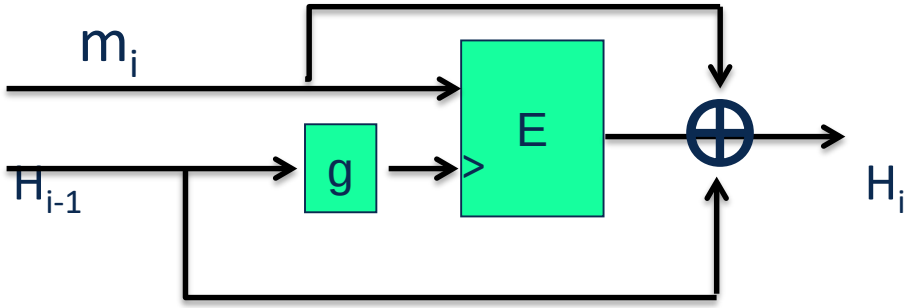
Equivalent to birthday attack → as good as possible.

Further variations

Matyas-Meyer-Oseas



Miyaguchi-Preneel



… and many insecure one‘s..

Using Block Ciphers in Practice

The efficiency of Davies-Meyer compression function in the Merkle-Damgård transformation depends on the efficiency of the chosen block cipher

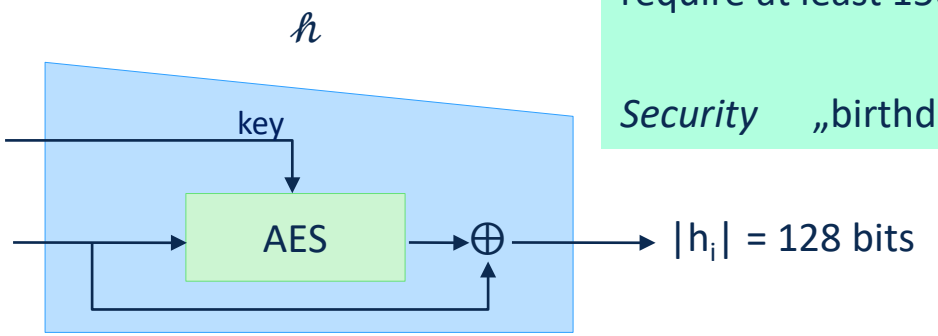
Example

AES-128 has key length κ = 128 bits, block length b = 128 bits
AES-192 has key length κ = 192 bits, block length b = 128 bits
AES-256 has key length κ = 256 bits, block length b = 128 bits

Davies-Meyer with AES

$|m_i|$ = 128 bits
$|m_i|$ = 192 bits
$|m_i|$ = 256 bits

$|h_{i-1}|$ = 128 bits



$|h_i|$ = 128 bits

Efficiency hashing of a 5MB file would require at least 156250 ≈ 2^{17} AES executions

Security „birthday attack“ $O(2^{64})$

block ciphers with much larger key lengths and block sizes needed…

End of Interlude, back to MACs!

So can we use these hash functions directly as a MAC?

Quick answer: *no, we need some secret (recall: CRC)!*

Intermediate answer is *„for special cases, yes“:*

Assume a public repository of files

and a ***public read-only space*** with their hashes

User can verify validity of the contents of a packet

Adversary cannot forge packets for given $H(F_i)$ (collision!)

Creating secure MACs, a first attempt

MAC: signing alg. **S(k,m)⟶t** and verification alg. **V(k,m,t) ⟶0,1**

First Idea: keyed hash functions, MAC: H(k||m)

Recall: Secure hash is collision resistant
But a secure MAC needs to be ***unforgeable*** under ***chosen message attack***
no existential forgery: no valid tag forged for arbitrary message

Consider Merkle-Damgard construction: s = H(m|| PB)
Can you forge a valid tag?

Feasible chosen message attack:

A – [m] –> C $\quad s = h(k||m||PB)$
C <– [s] – A

$s' = h(m||m') = h(s||m'||PB')$

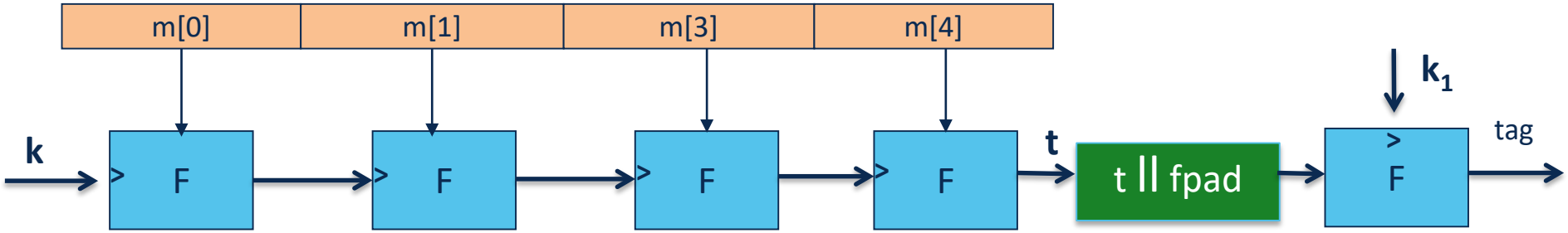
A – (m||m', s') –> C *and wins the game!*

Creating MACs using known PRF/PRPs: NMAC

Recall Merkle-Damgard, *k* instead of fixed IV:

Let $F: K \times X \longrightarrow X$ be a PRF, define new PRF $F_{NMAC}: K^2 \times X^{\leq L} \longrightarrow X$

Cascade:



Why the last encryption with second key?

Otherwise: cascade(k, m||m‘) = cascade(cascade(k,m)||m‘)

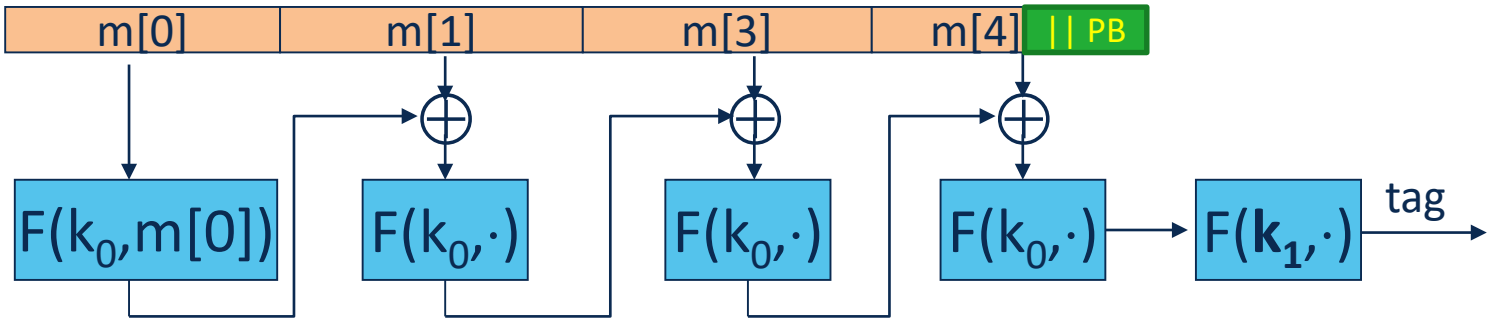
Extend raw CBC to CBC-MAC / ECBC

Recall Merkle-Damgard, PRFs & mode of operation

Let‘s create a ***MAC*** from AES (a PRP for small messages)!

Let $F: K \times X \longrightarrow X$ be a PRF, define new PRF $F_{ECBC}: K^2 \times X^{\leq L} \longrightarrow X$

CBC-MAC (ECBC: Encrypted Last Block CBC)



CBC-MAC insecure if |m| ≠ multiple of block size

Solve by padding (cmp. hash): PB = 10…0 (Why not “0…0”)?

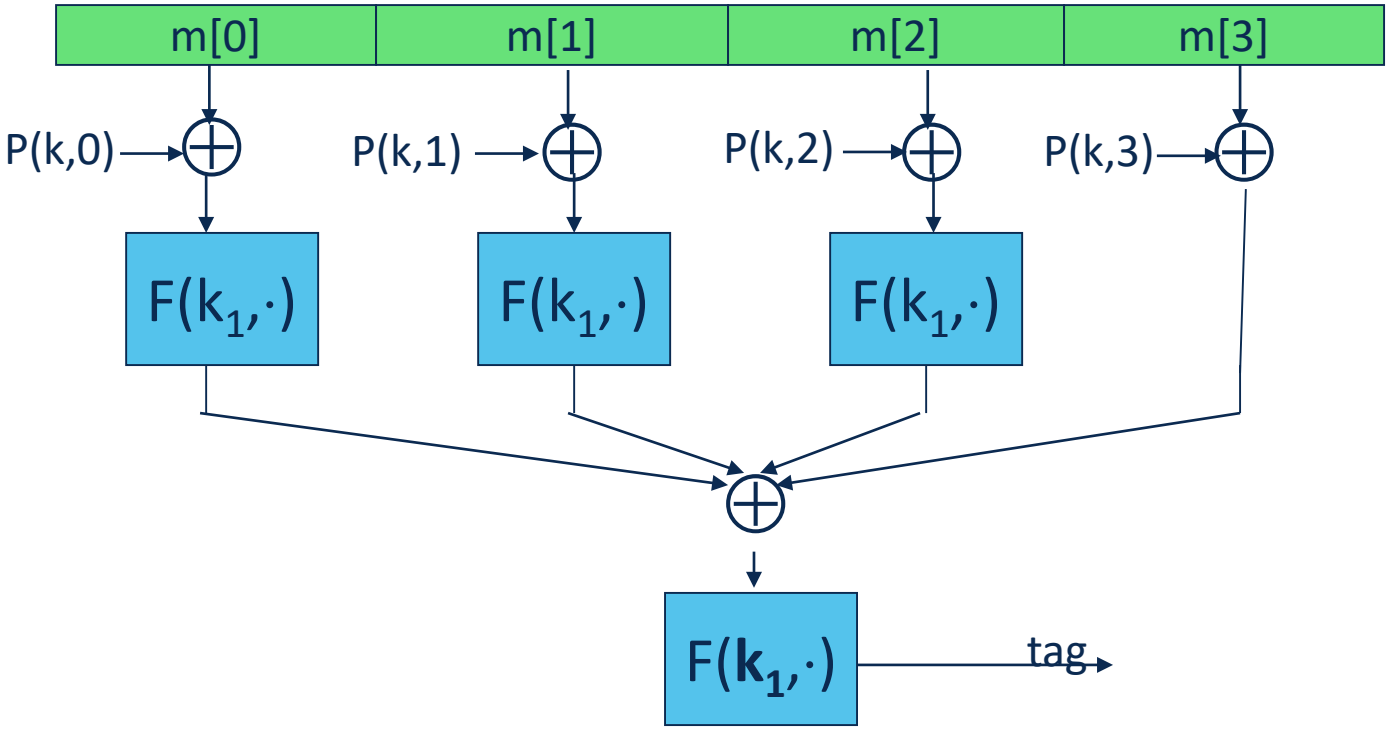
What if |m| = multiple of block size?

Speeding things up… PMAC

Let $F: K \times X \longrightarrow X$ be a PRF

Given P(k,i) being a function that is easy to compute, and key: (k, k_1)

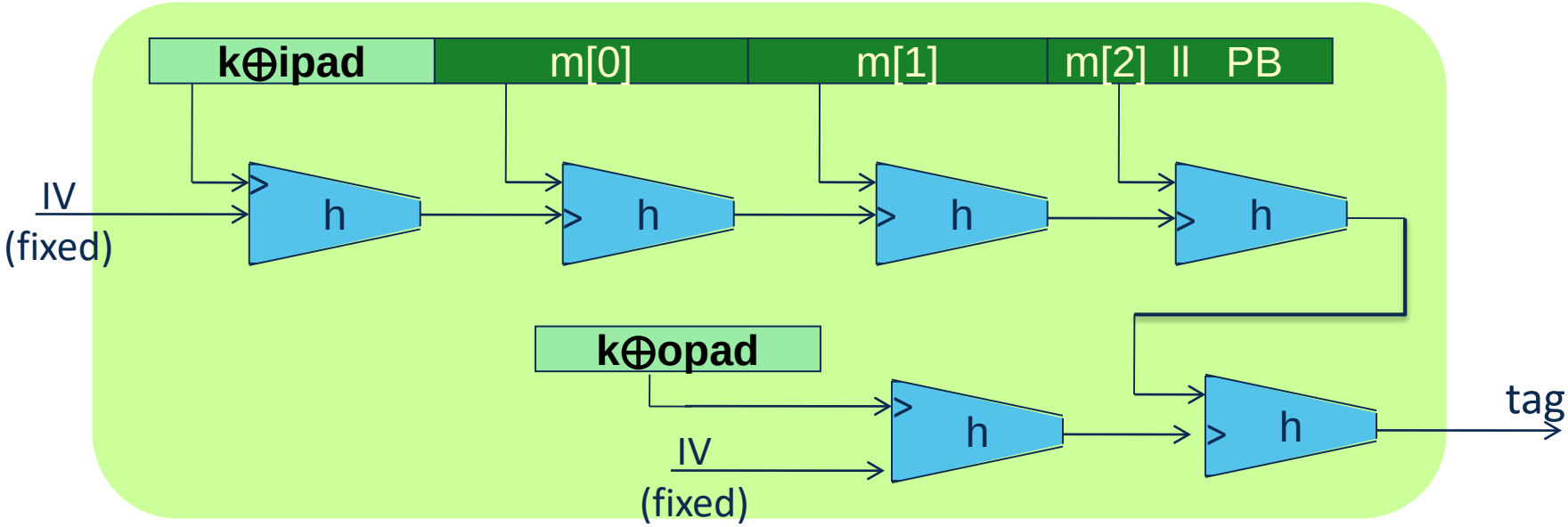
Define new PRF $F_{PMAC}: K^2 \times X^{\leq L} \longrightarrow X$



And finally: The HMAC (RFC 2104)

Hashing is fast, but H(k||m) insecure

Solution: encase message with keys! (Essientially double NMAC)



HMAC: $S(k, m) = H(\ k \oplus opad \ ||\ \mathbf{H(\ k \oplus ipad \ ||\ m\ )}\ )$

(used in TLS, IPsec,...)

Other Approaches to MACs and Hashing

Merkle-Damgard on compression functions

- Create an H for arbitrary sized input to fixed size output
- Using „small“ compression functions
- We already know (and have implemented) some small compression functions, which we seem to „understand“ well (we have a good feeling about AES, and it‘s already implemented in hardware)
- Given the above, we can prove the security of H

However: other approaches to create secure arbitrary length hash functions do exist…

Interlude: Arbitrary Length Output

Recall the definition of $h: M \rightarrow S$ with $M = \{0,1\}^*$ and $S = \{0,1\}^s$

Proving security: the output is indistinguishable from an output of a really *random oracle* $RO: M \rightarrow Y$ with $Y = \{0,1\}^s$

Now what happens if $F: M \rightarrow Y$ with $Y = \{0,1\}^*$

The security in the random oracle model depends on $|S|$, does the security of the latter increase with the length of the output?

Probably not, rather with the size of internal state…

SHA-3 Competition

2005: NIST discusses new standard for Secure Hash Algorithm
rationale: all prior standards are based on MD, consider what happened if there was a systematic break…

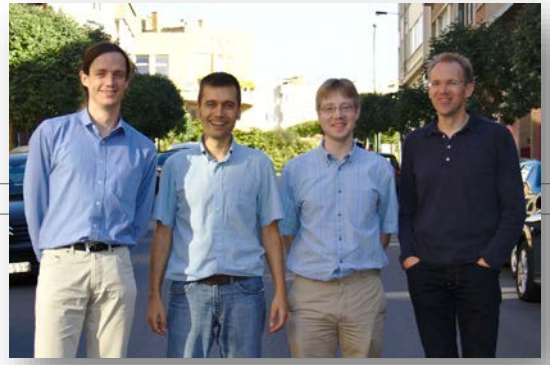
2007: Call for proposals; requirements (replacement for SHA-2):
224, 256, 384, and 512 bit output length

2008: 64 submissions

2009, 2010, 2012: Candidate conferences (14 candidates, 5 finalists:
BLAKE (Aumasson, Henzen, Meier, Phan, based on ChaCha20), **Grøstl** (TU Graz, DTU, based on AES), **JH** (Hongjun Wu), **Keccak** (Bertoni, Daemen, Peeters, van Assche), **Skein** ("Team World" including Schneier, Lucks, based on Threefish).

2012: Announcement of Keccak as SHA-3

Keccak – SHA3

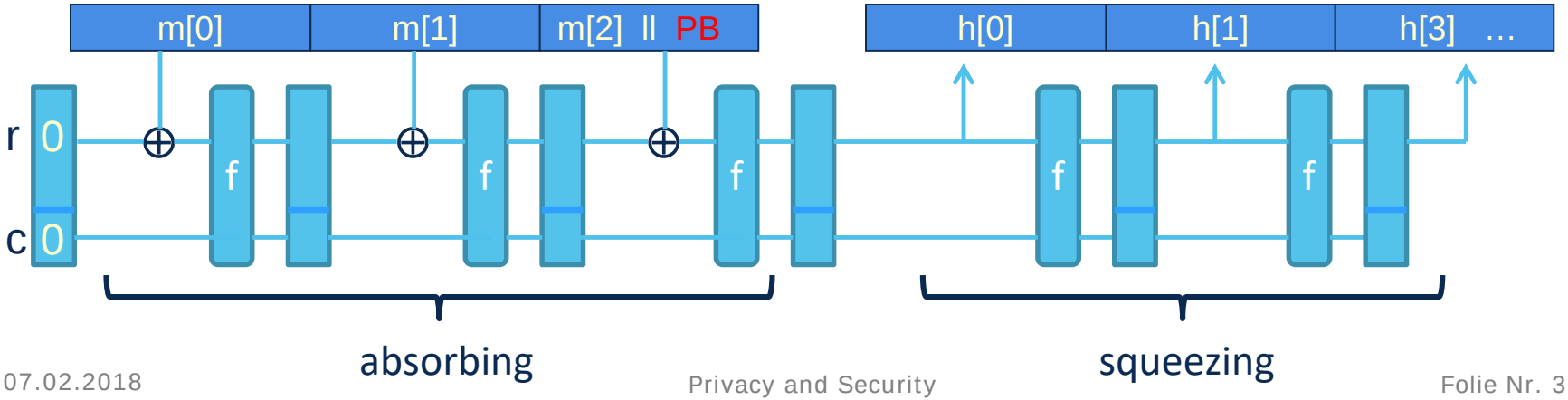


Specification of two modes:

- SHA-2 replacement: fixed length output of 224, 256, 384, 512 bits
- Variable length output: output of arbitrary length
- Keccak[r,c] with internal permutation Keccak-f[b]

Construction of two phases

- Absorb: block-wise input of message
- Squeeze: output of required bits as hash value



Keccak State – the Sponge

Keccak[r,c] parameters:

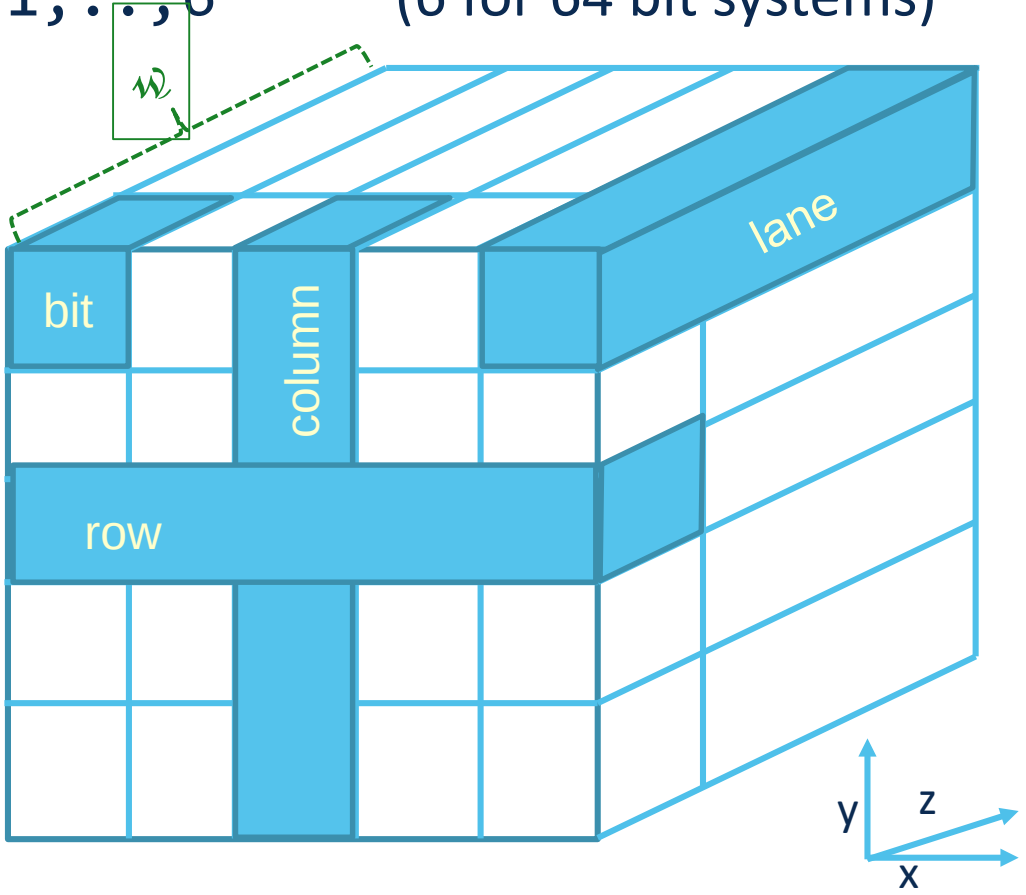
- bit rate r, capacity c (c = 256,512)
- word length w = 2^l l=0,1,..,6 (6 for 64 bit systems)

b = r+c = 5x5xw

= *25,50, 100*

100,200,400

800,1600

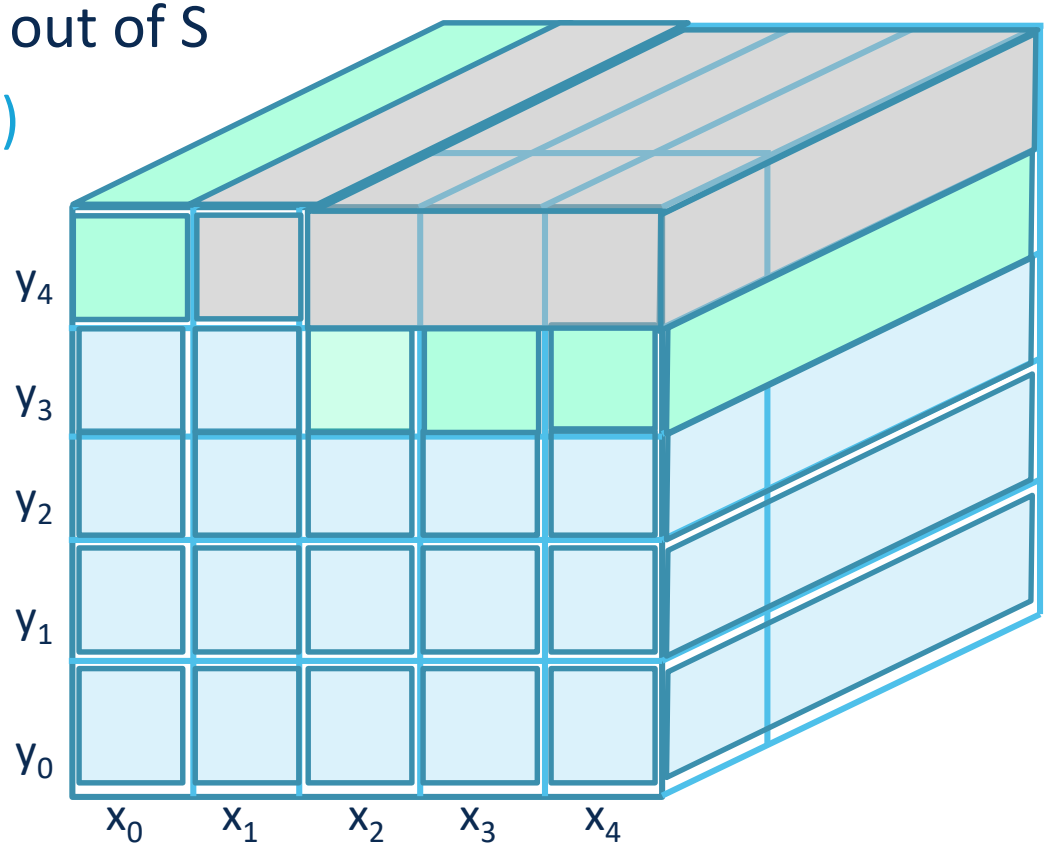


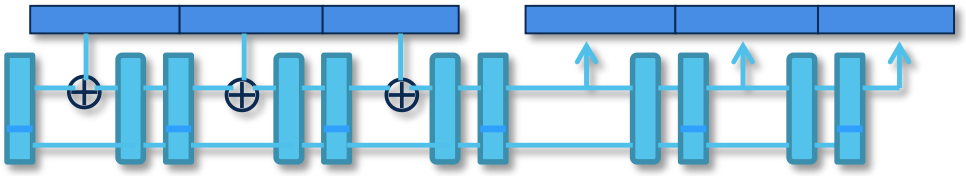
Keccak Bit Rate and Capacity

Absorbing into and squeezing out of S

- 256 bit security (c=512 bits)
- 128 bit security

Bits in c never in- nor output



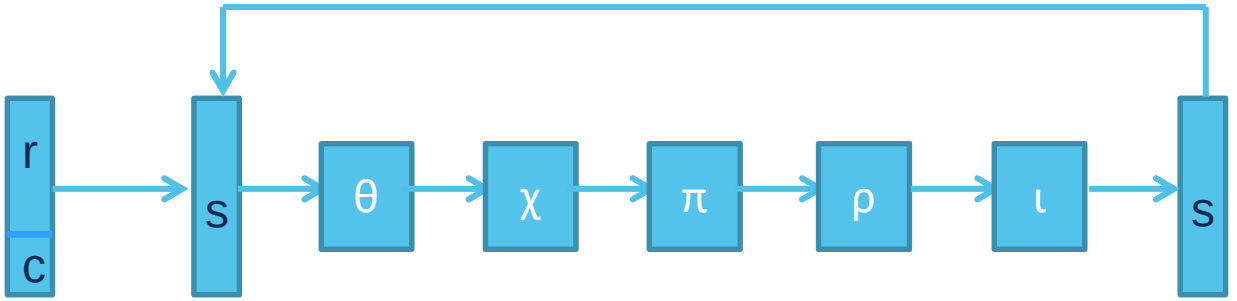


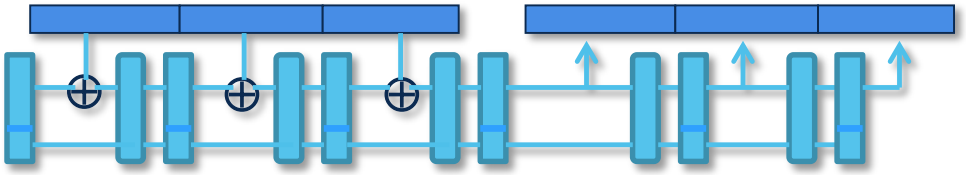
Permutation in Several Rounds

Keccak permutes state in 12 + 2l rounds

- *32 bit processor -> Keccak-f[800] -> 22 rounds*
- *64 bit processor -> Keccak-f[1600] -> 24 rounds*
- *(Keccak-f[25] -> 12 rounds)*

Five operations for each round:



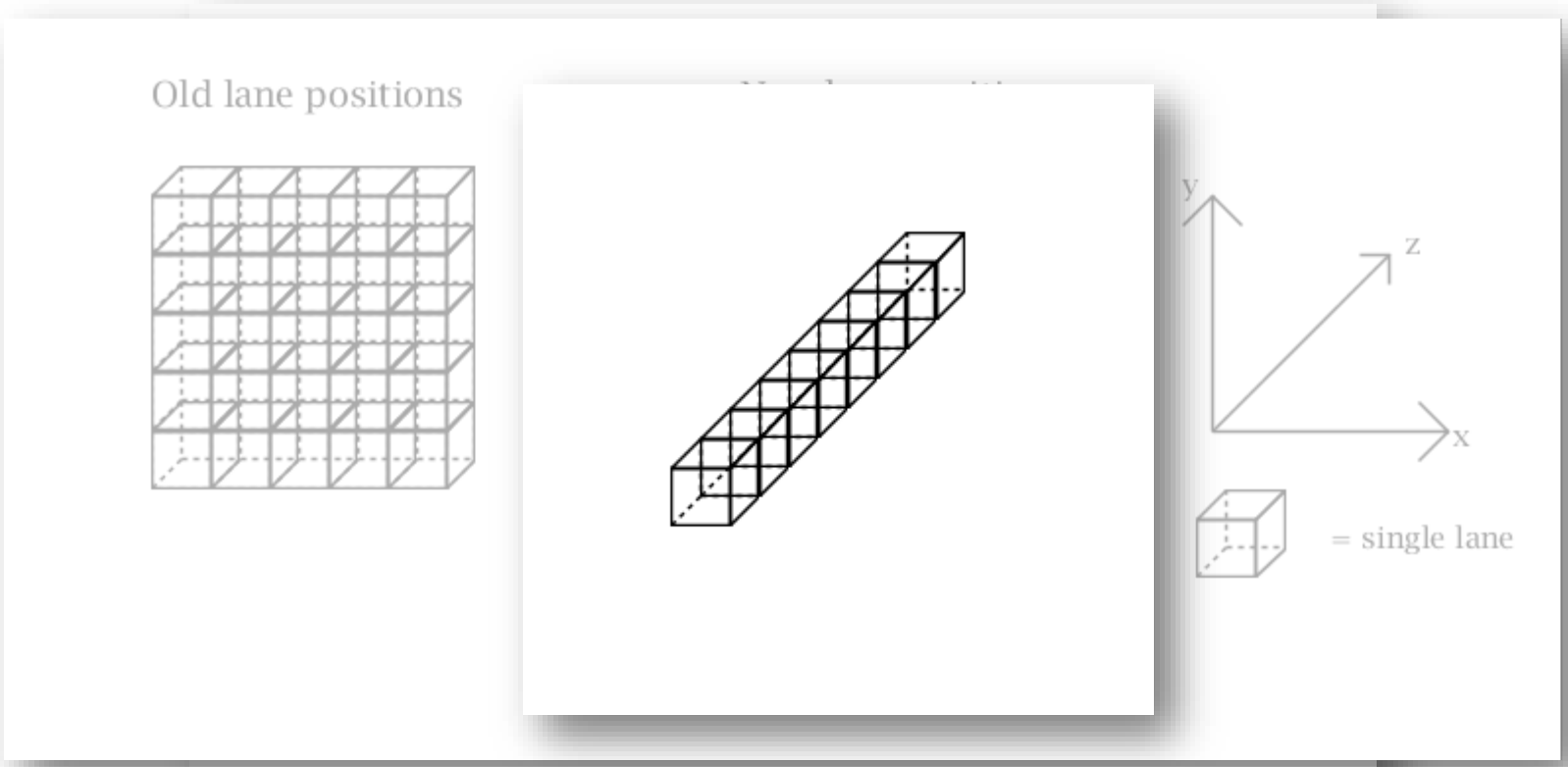


The Keccak-f Permutation

http://celan.informatik.uni-oldenburg.de/kryptos/info/keccak

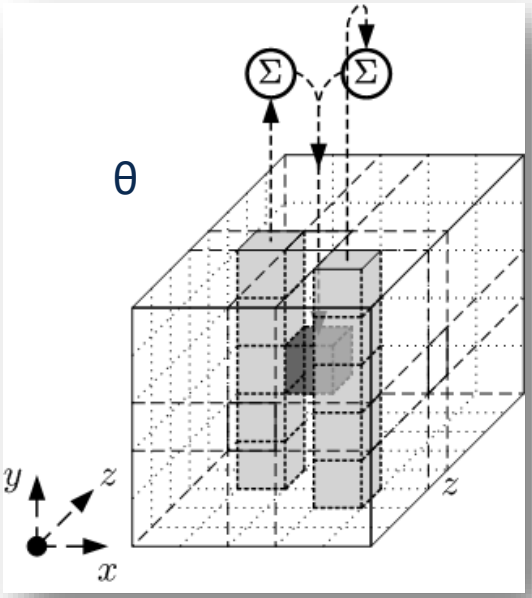
Internal function defined as permutation Keccak-f[b]:

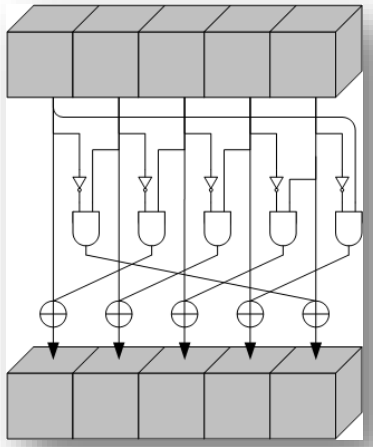
$$S \leftarrow \iota \circ \rho \circ \pi \circ \chi \circ \theta\,(S)$$



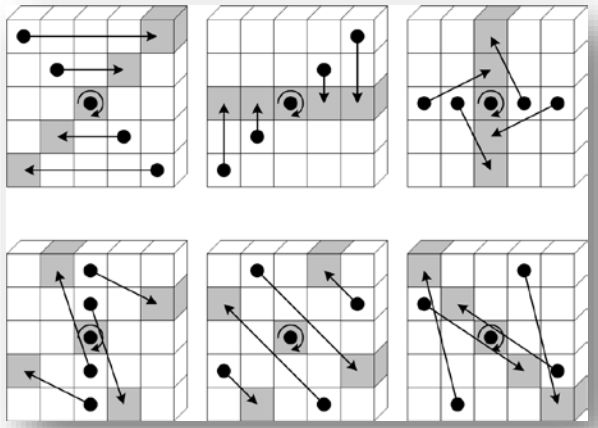
- Iota adds predefined constant to lane [0,0]

The Keccak-f Permutation

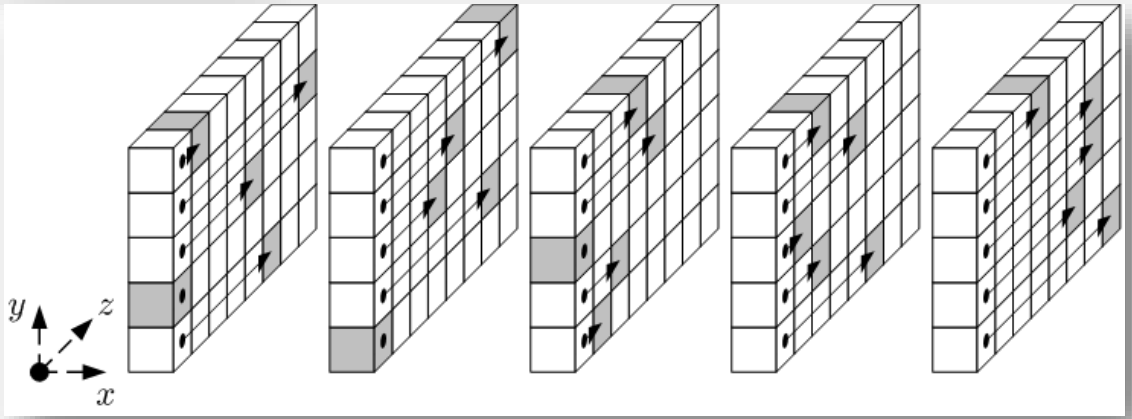




χ



π

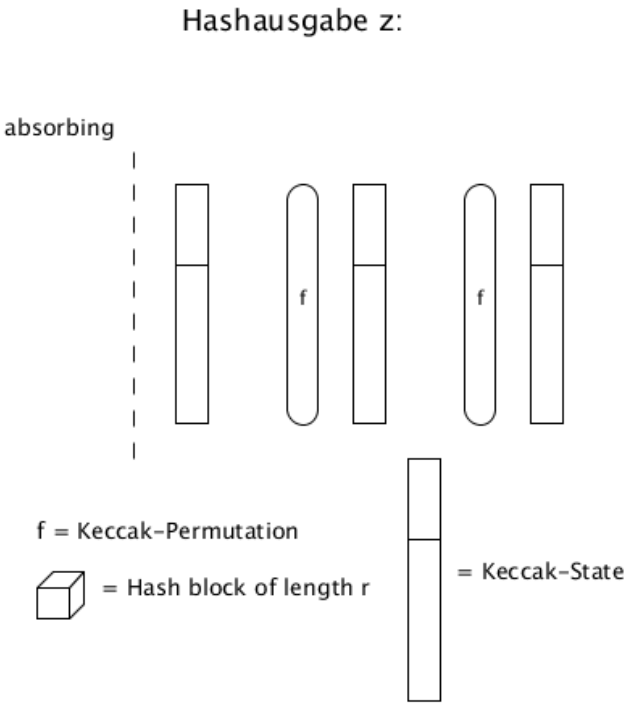


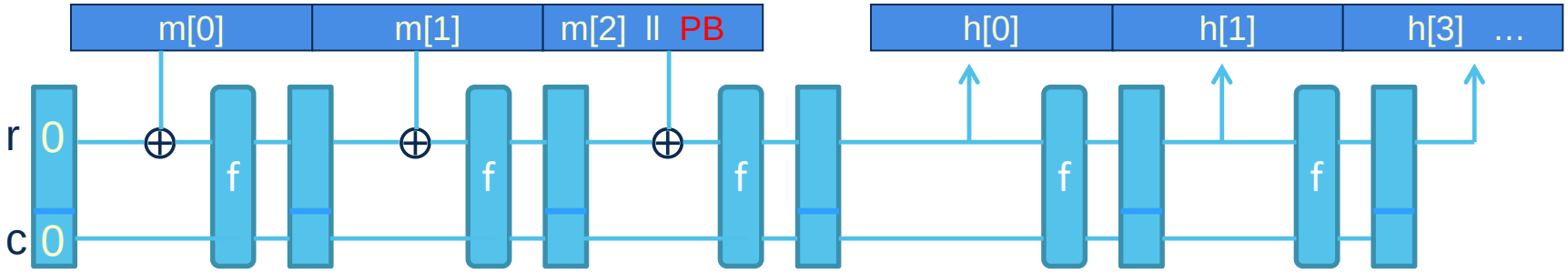
ρ

http://keccak.nokoen.org

Kezzak – Squeeze

http://celan.informatik.uni-oldenburg.de/kryptos/info/keccak





Using Keccak as SHA3 (and SHAKE)

Keccak[r,c] (l=6 -> Keccak-f[1600]) standardized as SHA-3:

Algorithm	Output size	Block size	Bit security
SHA3-224	224	1152	112
SHA3-256	256	1088	128
SHA3-384	384	832	192
SHA3-512	512	576	256
SHAKE128	d	1344	128*
SHAKE256	d	1088	256*

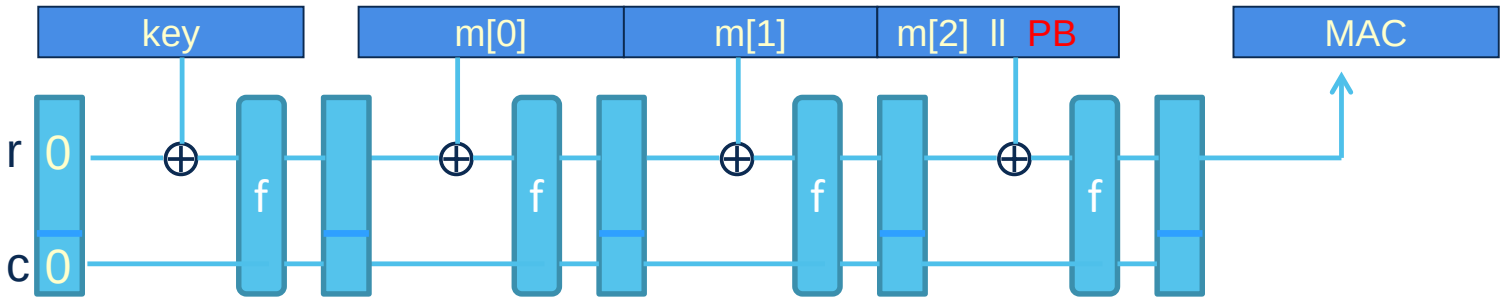
* or d/2, if smaller

Using SHA3 and SHAKE for Integrity

Can we implement a secure MAC as:

SHA-3(k||m)?

How?

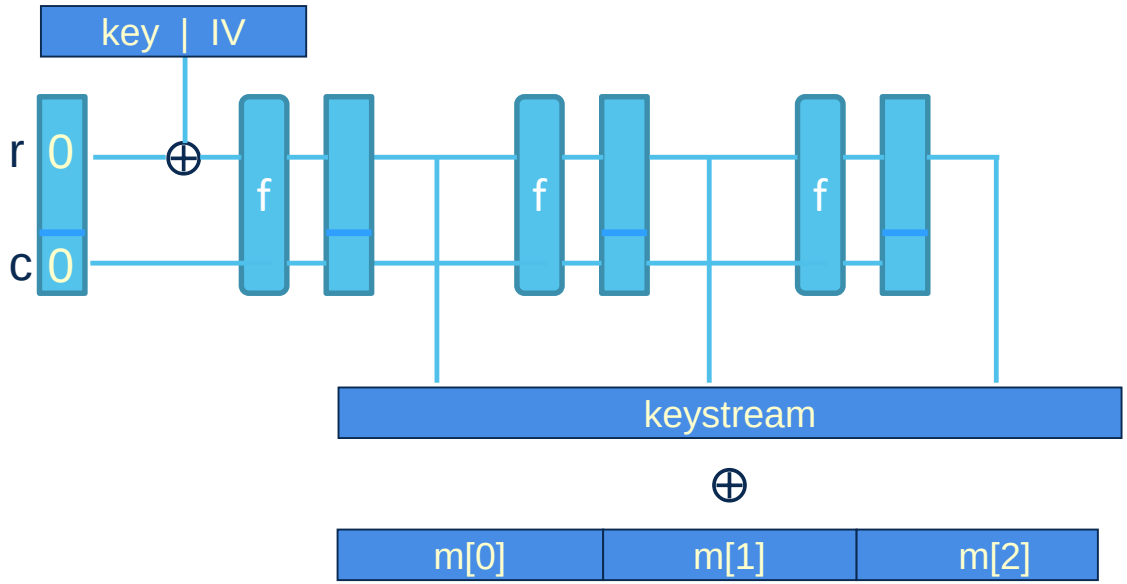


Why?

Using SHA3 and SHAKE for Confidentiality

Can we implement a stream cipher with SHA3?

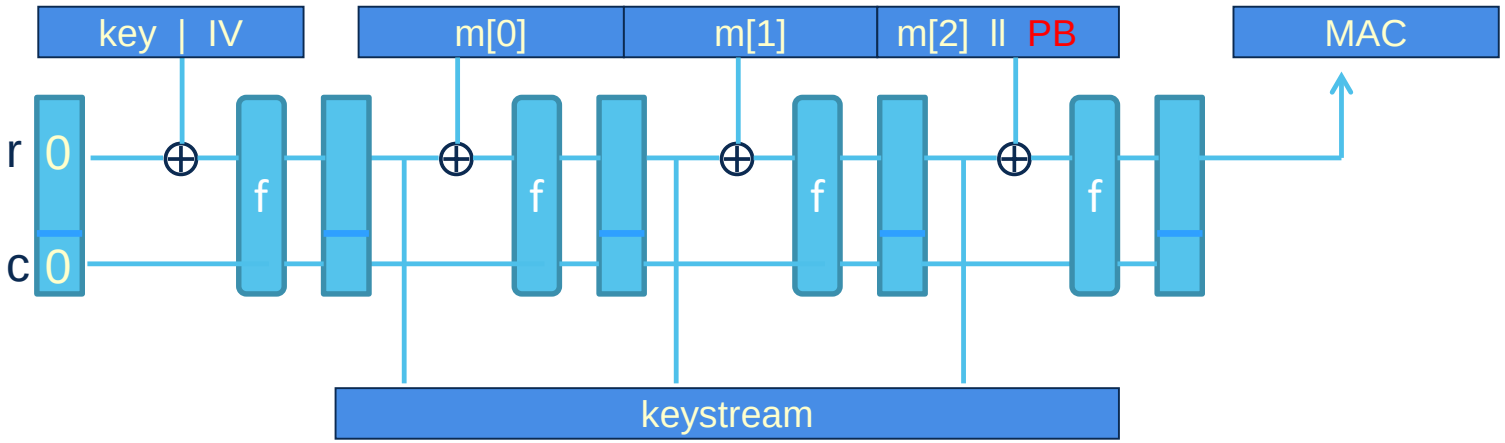
How?



SHA3 for Confidentiality and Integrity

Stream ciphers are malleable… Can we achieve authenticated encryption with SHA3?

How?



Can we be even faster: One time MAC

Can be secure against **all** adversaries and faster than previous MACs

Let q be a large prime (e.g. **$q = 2^{128}+51$**) *(or: 130-5)*

key = (a, b) ∈ $\{1,\ldots,q\}^2$ (two random ints. in [1,q])

msg = (m[1], …, m[L]) where each block is 128 bit int.

$S(\text{key}, \text{msg}) = P_{msg}(a) + b \pmod q$

where $P_{msg}(x) = \sum_{i=1}^{L} x^i \cdot m[i]$

$= m[L]\cdot x^L + \ldots + m[1]\cdot x$ is a poly. of deg. L

One-time MAC -> Many-time MACs

Let (S,V) be a secure one-time MAC over $(K_I, M, \{0,1\}^n)$.

Let $F: K_F \times \{0,1\}^n \longrightarrow \{0,1\}^n$ be a secure PRF.

slow but short inp — fast long inp

Carter-Wegman MAC: $CW((k_1,k_2), m) = (r,\ F(k_1,r) \oplus S(k_2,m))$

for random $r \longleftarrow \{0,1\}^n$.

Thm: If (S,V) is a secure **one-time** MAC and F a secure PRF then CW is a secure MAC outputting tags in $\{0,1\}^{2n}$.

Timing Attacks on MACs

A *(badly implemented)* verification oracle

- Compares output of MAC byte by byte
- *(and returns as soon as a byte differs)*

Mallory has access to verification oracle, aims at universal forgery

?	?	?	?	?		?	?	?	?	?
B_1	B_2	B_3	B_4	B_5		B_{12}	B_{13}	B_{14}	B_{15}	B_{16}

Bonus Take-Away:
Too much speed (performance) can hurt…

(Given arbitrary message (pirated code), guess correct MAC)

1. For B_i try all 256 possible values
2. Measure time until rejection (comparison per byte 2.2ms)

For 16 byte MAC (2^{128}), how many attempts do you need?

Concluding: Security through MACs

MACs verify integrity of messages

S(k,m) ; V(k,m,t) → secret key must be used, known to verifier

MAC hard to forge without secret key, but *integrity purely mutual*:

- Once key is disclosed, receiver can create arbitrary new tags!
- ⇒ Proof of origin not towards third parties (no non-repudiation!)

But to achieve this, we first have to understand keys better…

Summary

You can explain the goals and ideas of message integrity

You know different adversary and security models for MACs

You have seen different constructions of hash functions

You specifically can explain the Merkle-Damgard construction, MD5, SHA-1

You know how to create collisions (and why that‘s bad)

You can explain how to create secure MACs from hash functions and PRFs

You can construct and explain the details of CBC, and HMAC