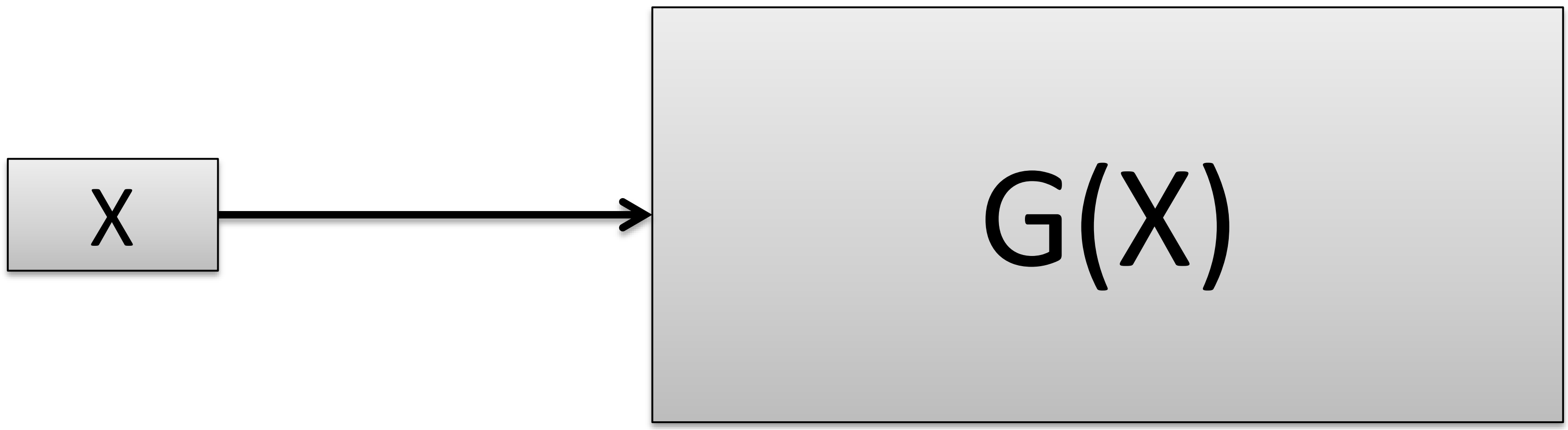
Insomni'Hash

Jean-Philippe Aumasson

NAGRA KUDELSKI

1 0 1 1 0 1 1 1 0 0 1 0 0 0

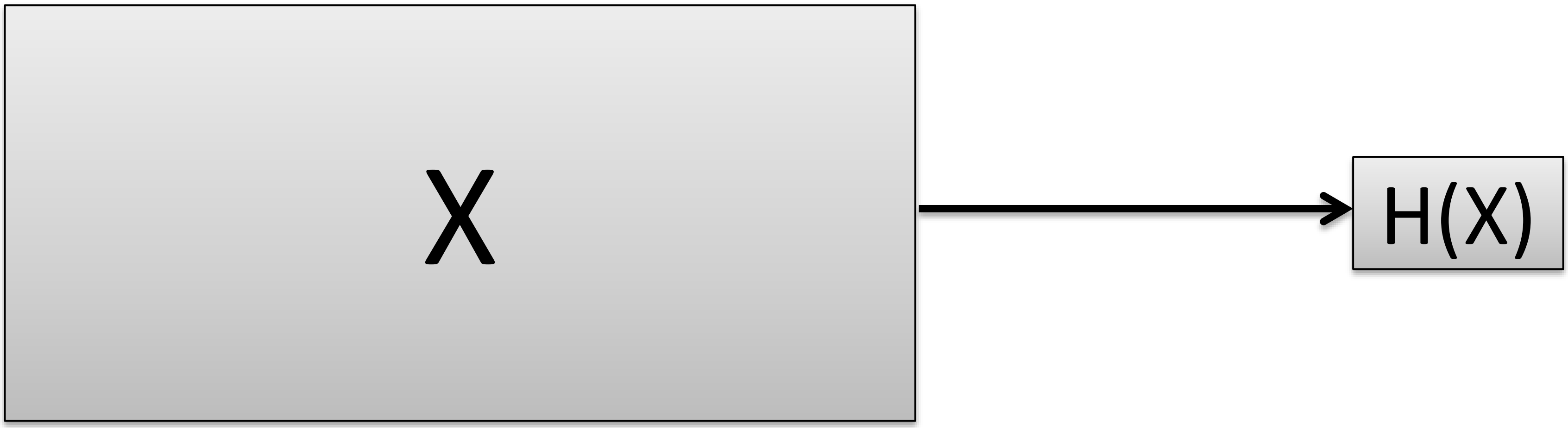
Générateur pseudoaléatoire



X de (petite) longueur fixe n

G(X) de longueur arbitraire

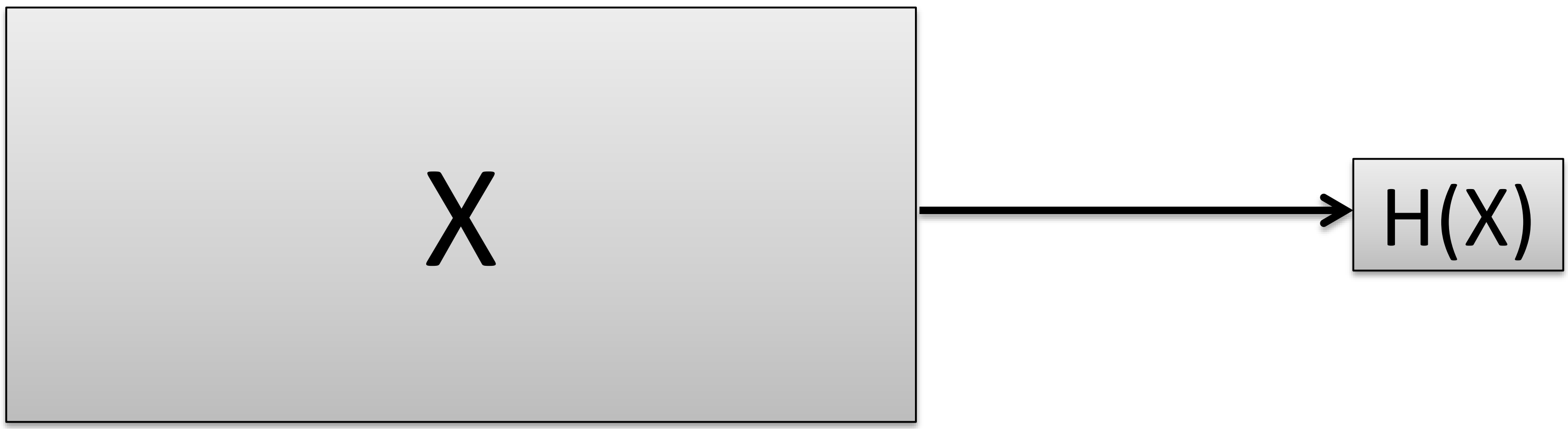
Hachage



X de longueur arbitraire

H(X) de (petite) longueur fixe n

Hachage **cryptographique**



X de longueur arbitraire

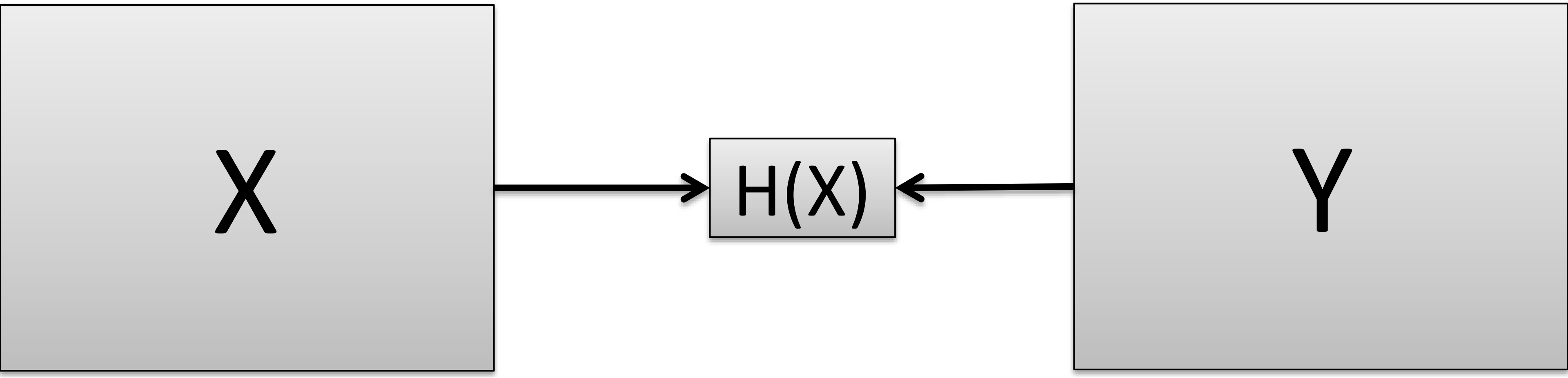
H(X) de (petite) longueur fixe n

Secure

Secure?

Résistance aux **collisions**

$$H(X) = H(Y),\ X \mathrel{!=} Y$$



Avec $2^{n/2}$ messages on a environ

$$(2^{n/2} \times 2^{n/2}) / 2 = 2^{n-1}$$

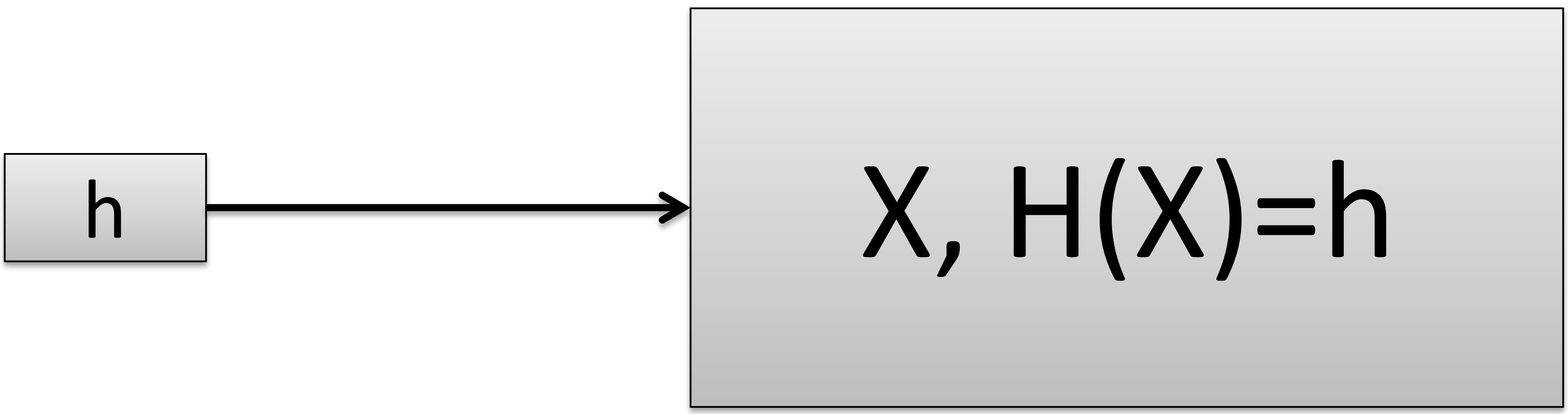
paires (X, Y) candidates

Une paire a une prob. $1/2^n$ de collision

Une liste triée de $\approx 2^{n/2}$ (X, H(X))'s suffit à trouver une collision (aussi sans mémoire)

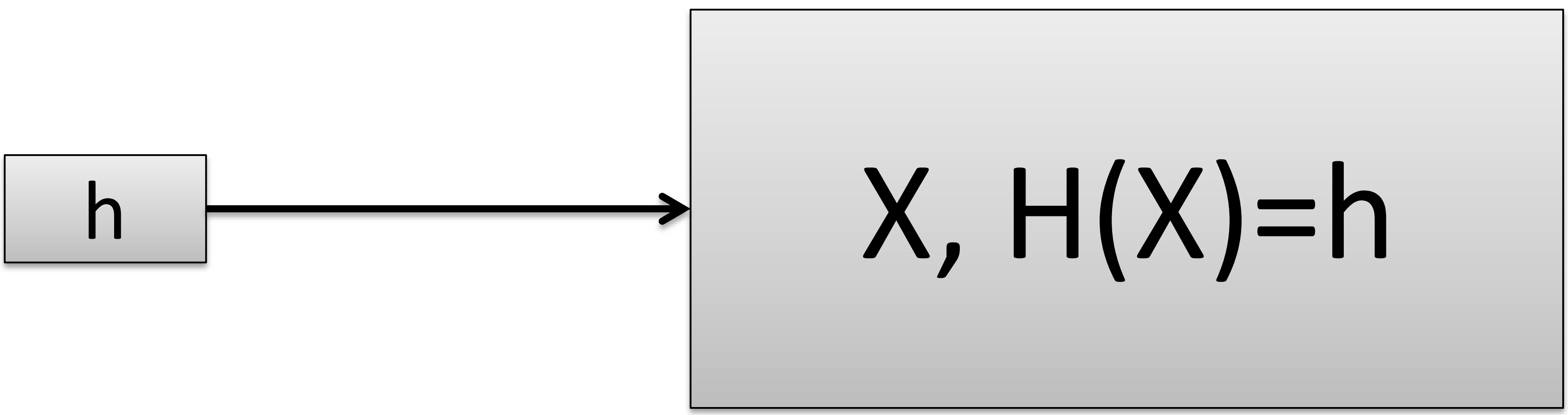
Résistance aux **préimages** (v1)

Pour un h aléatoire



Résistance aux **préimages** (v1)

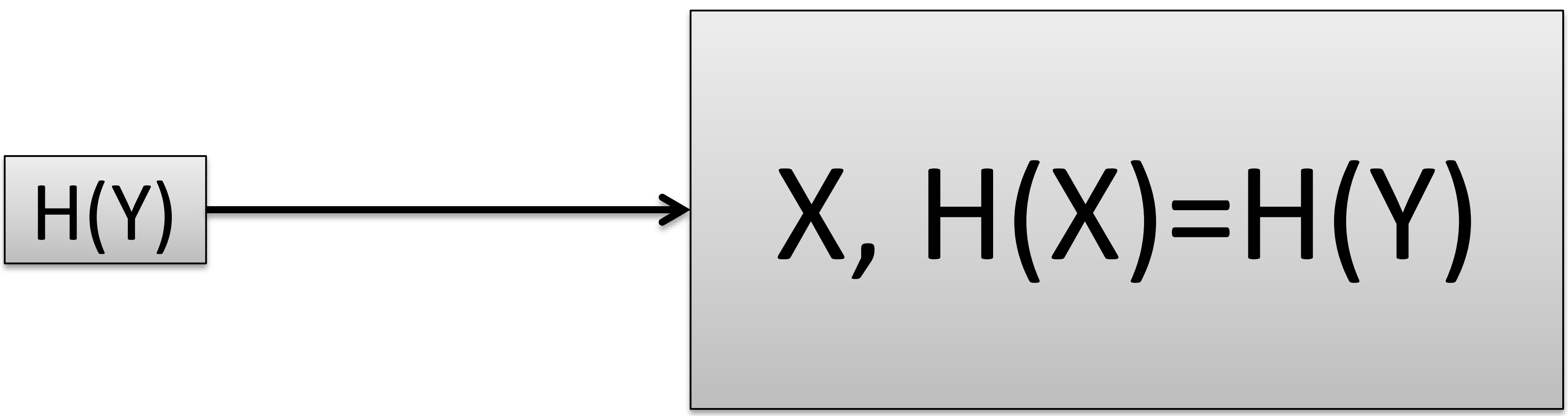
Pour un h aléatoire



Ex: H tel que pour tout X, H(X)=0

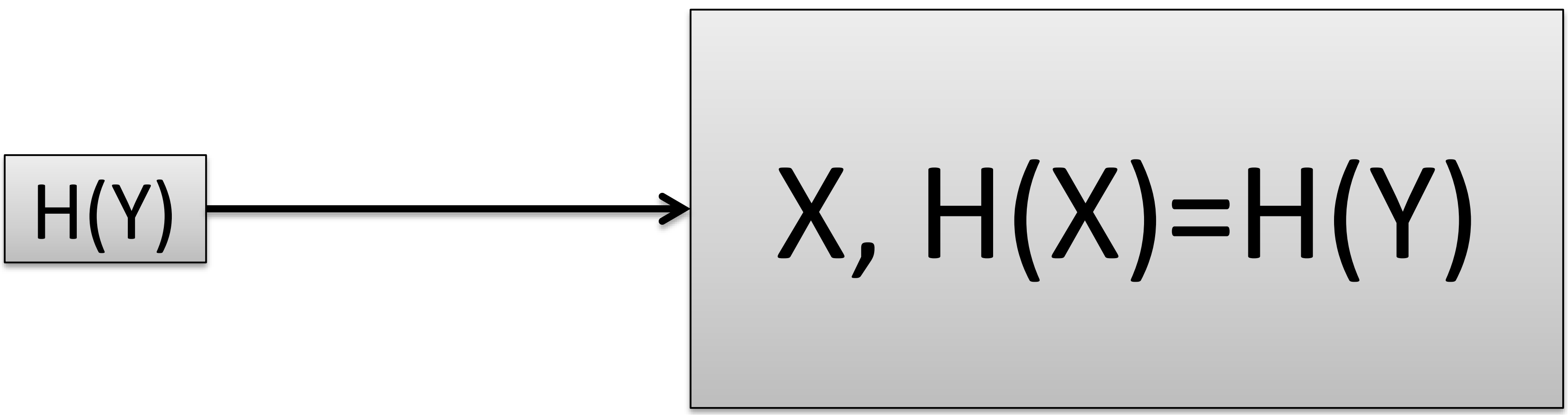
Résistance aux **préimages** (v2)

Pour un Y aléatoire et inconnu



Résistance aux **préimages** (v2)

Pour un Y aléatoire et inconnu



Ex: H(X)=0 si |X|<128, H(X) aléatoire sinon

Résistance aux **préimages**

H est “one-way” ;-)

=> 2^{n-1} essais H(X) en moyenne

Résistance aux **secondes préimages**

Pour un Y aléatoire donné

H(Y) → X, H(X)=H(Y), X!=Y

Ex: H(X)=0 si |X|<128, H(X) aléatoire sinon

1 0 1 1 0 1 1 1 0 0 1 0 0 0

Pseudorandom function (PRF)

Pour une clé secrète K, $H_K()$ doit être **indistinguable** d'une fonction aléatoire

Concrètement: on ne doit pas pouvoir prédire $H_K(X)$ sans connaître X

Pseudorandom function (PRF)

+ Mathématiquement rigoureux

+ Aucune des “failles” précédentes

- Considère H comme une black-box

- Nécéssite une clé secrète

42

Un H “secure” doit être

Indistinguable d’une fonction parfaite

Un H “secure” doit être

Indistinguable

d’une fonction **aléatoire**

Un H “secure” doit être

Indistinguable

d’un **oracle aléatoire**

Un H “secure” doit être

Indifférentiable

d’un **oracle aléatoire**

Une définition “simple”

Definition 1 ((Statistical, Strong) Indifferentiability). *Let $q, \sigma : \mathbb{N} \to \mathbb{N}$ and $\epsilon : \mathbb{N} \to \mathbb{R}$ be three functions of the security parameter n. A construction $\mathcal{C}$ with oracle access to an ideal primitive $\boldsymbol{F}$ is said to be statistically and strongly (q, σ, ϵ)-indifferentiable from an ideal primitive $\boldsymbol{G}$ if there exists an oracle ITM $\mathcal{S}$ such that for any distinguisher $\mathcal{D}$ of total oracle queries cost at most q, $\mathcal{S}$ makes at most σ oracle queries, and the following holds:*

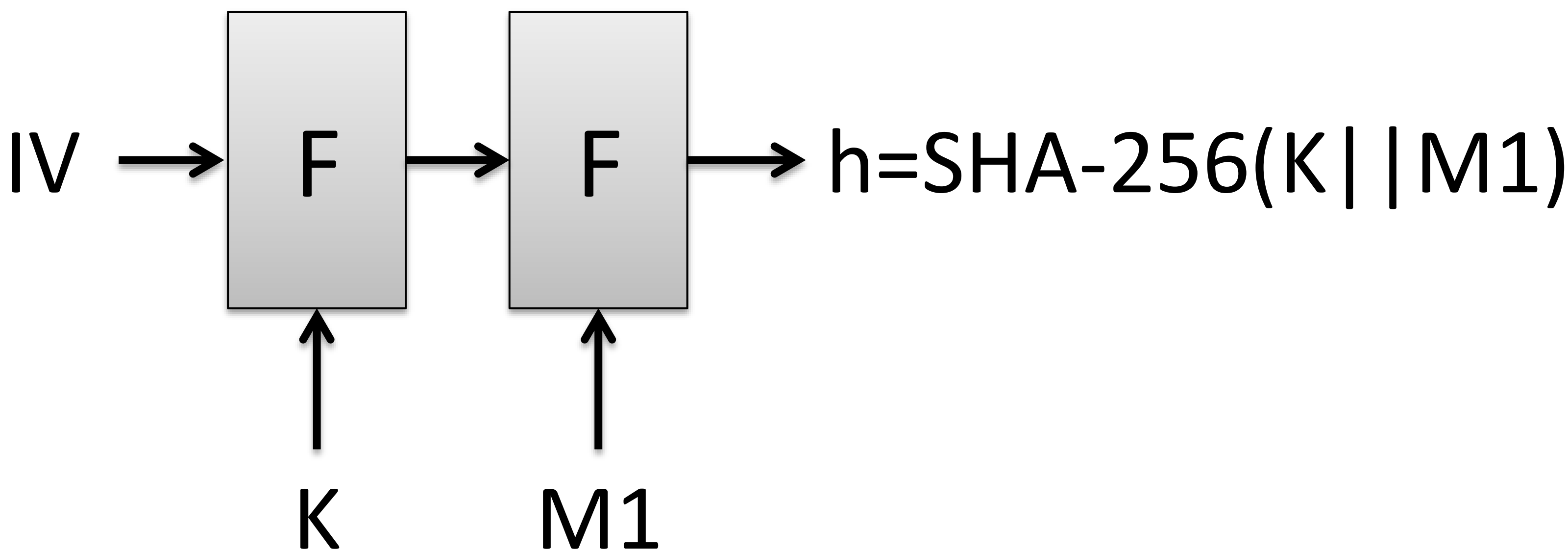
$$\left| \Pr\left[\mathcal{D}^{\boldsymbol{G}, \mathcal{S}^{\boldsymbol{G}}}(1^n) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{C}^{\boldsymbol{F}}, \boldsymbol{F}}(1^n) = 1\right] \right| \leq \epsilon \ .$$

$\mathcal{C}^{\boldsymbol{F}}$ is simply said to be statistically and strongly indifferentiable from $\boldsymbol{G}$ if for any $q \in \mathtt{poly}(n)$, the above definition is fulfilled with $\sigma \in \mathtt{poly}(n)$ and $\epsilon \in \mathtt{negl}(n)$.

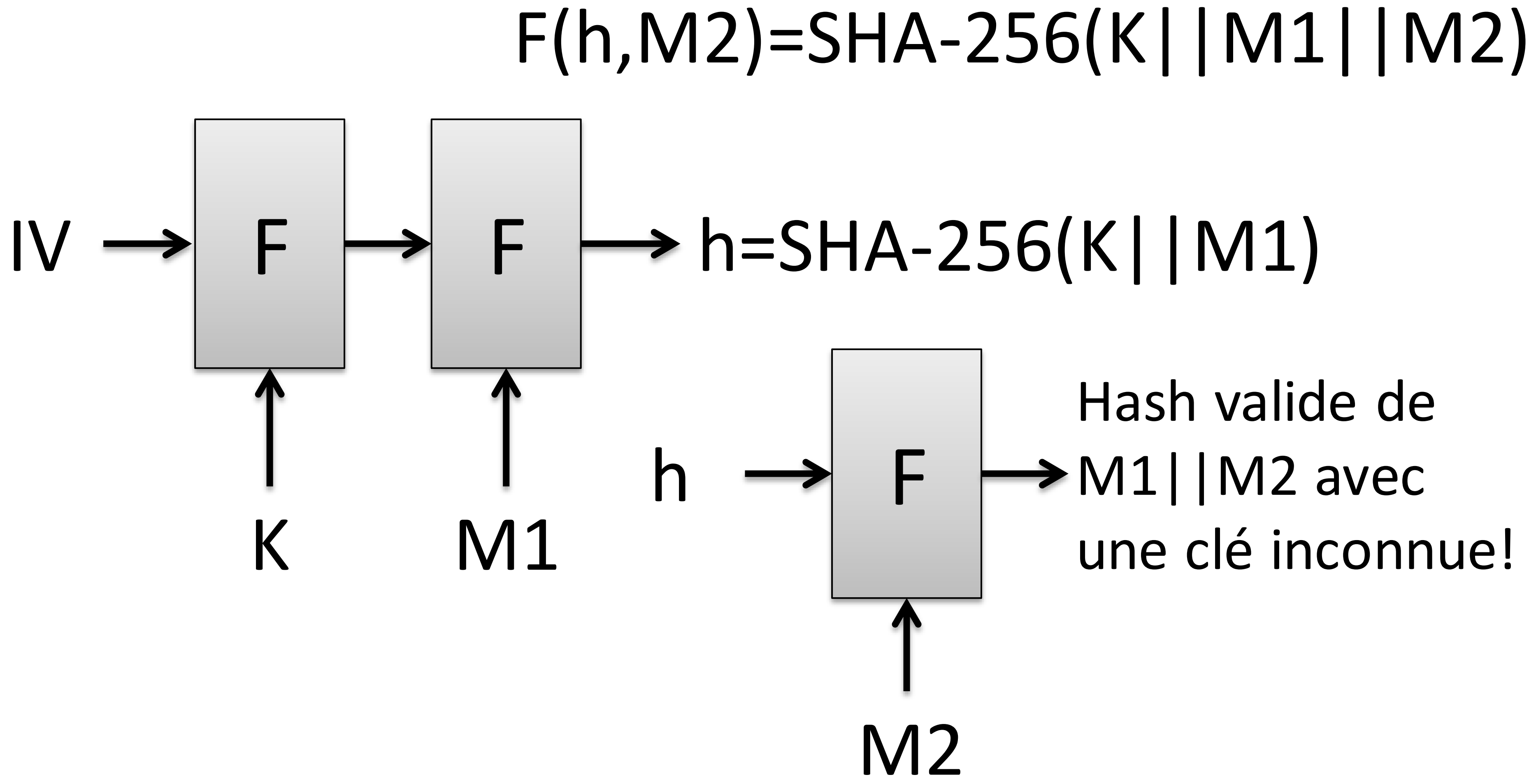
http://eprint.iacr.org/2011/496.pdf

SHA-256 n’est pas indifférentiable d’un oracle aléatoire

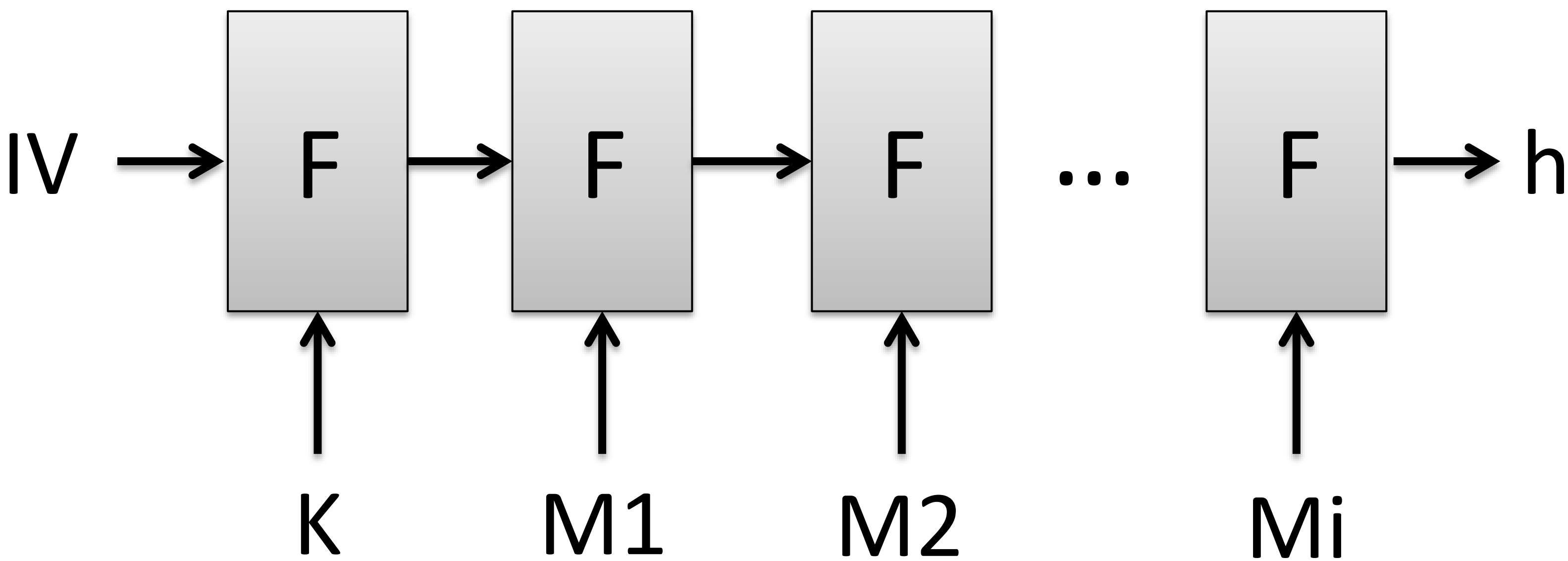
SHA-256 n’est pas indifférentiable d’un oracle aléatoire



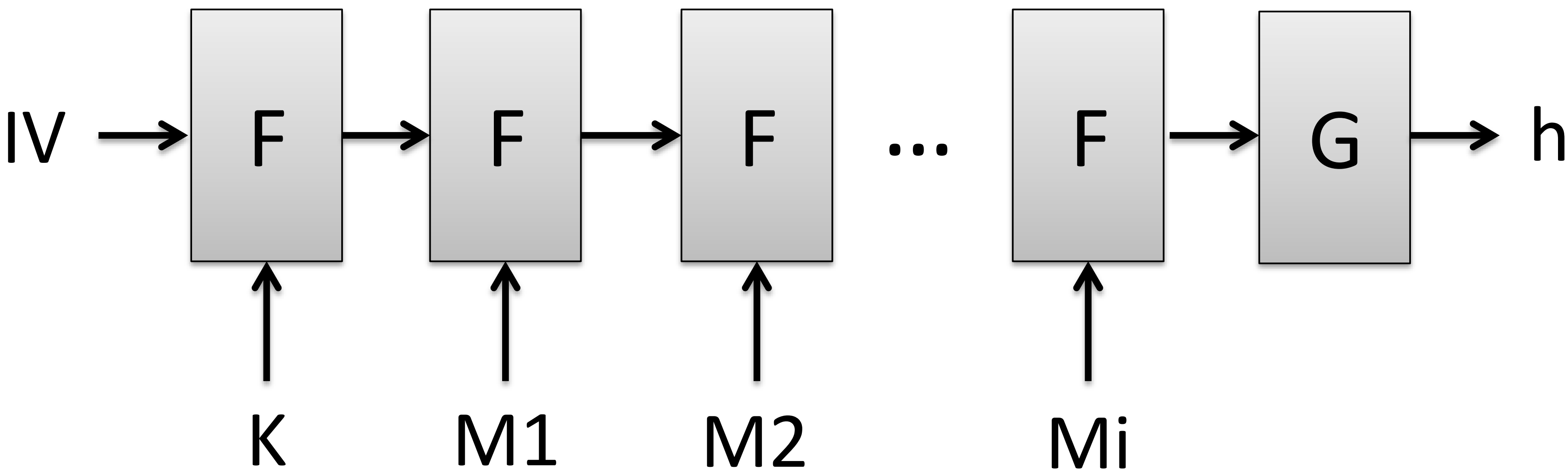
SHA-256 n'est pas indifférentiable d'un oracle aléatoire



“Insecure”



“Secure”



Indifférentiabilité: la notion ultime? Ou pas…

Careful with Composition: Limitations of Indifferentiability and Universal Composability

Thomas Ristenpart[*] Hovav Shacham[†] Thomas Shrimpton[‡]

June 22, 2011

Abstract

We exhibit a hash-based storage auditing scheme which is provably secure in the random-oracle model (ROM), but easily broken when one instead uses typical indifferentiable hash constructions. This contradicts the widely accepted belief that the indifferentiability composition theorem applies to *any* cryptosystem. We characterize the uncovered limitation of the indifferentiability framework by show-

EUROCRYPT 2011

</théorie>

<pratique>

Applications et notions de sécurité

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        05:e2:e6:a4:cd:09:ea:54:d6:65:b0:75:fe:22:a2:56
    Signature Algorithm: sha1WithRSAEncryption
    Issuer:
        emailAddress            = info@diginotar.nl
        commonName              = DigiNotar Public CA 2025
        organizationName        = DigiNotar
        countryName             = NL
    Validity
        Not Before: Jul 10 19:06:30 2011 GMT
        Not After : Jul  9 19:06:30 2013 GMT
    Subject:
        commonName              = *.google.com
        serialNumber            = PK000229200002
        localityName            = Mountain View
        organizationName        = Google Inc
```

sha1WithRSAEncryption(M)

= RSA(SHA-1(M))

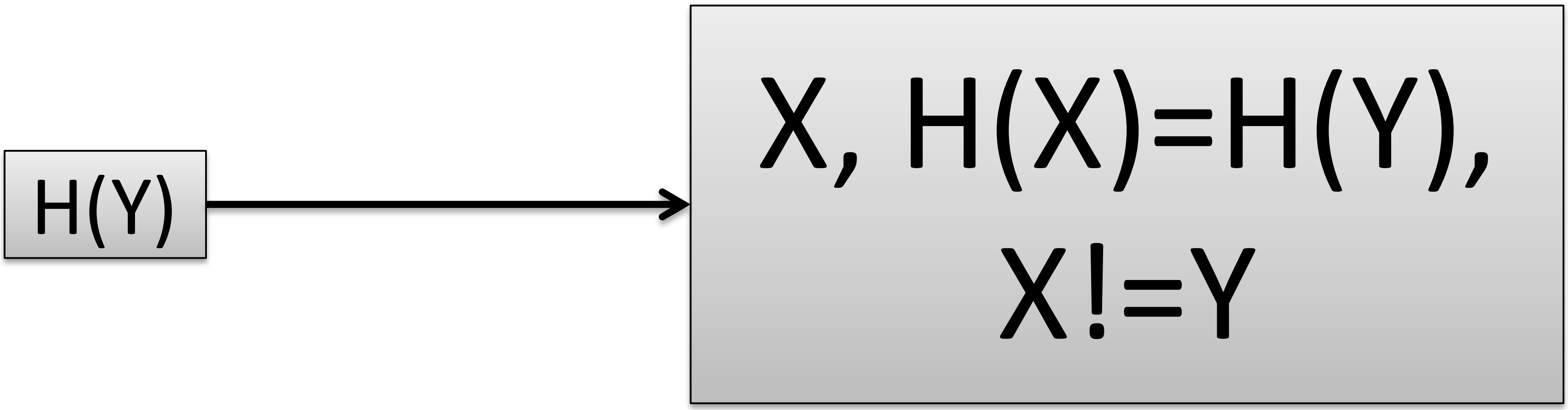
$$\text{signature}(M) = \text{sign}(H(M))$$

signature(M) = sign(H(M))

sign = RSA, DSA, ECDSA, etc.

H = SHA-1, SHA-256, etc.

But de l’adversaire: forger une signature valide pour un message encore jamais signé signature(M) = sign(H(M))



Seconde préimage = possibilité de forger une signature en bypassant l’algorithme “Sign”

$$signature(M) = sign(H(M))$$

Résistance aux collisions nécessaire?

$$\text{signature}(\ M\ ) = \text{sign}(\ H(\ M\ )\ )$$

Résistance aux collisions nécessaire?

$$H(X) = H(Y),\ \text{signature}(X) = \text{signature}(Y)$$

$$\text{signature}(\ M\ ) = \text{sign}(\ H(\ M\ )\ )$$

Résistance aux collisions nécessaire?

$H(X) = H(Y)$, $\text{signature}(X) = \text{signature}(Y)$

OUI, nécessaire à la non-répudiation

25C3 (CCC 2008)

Collisions exploitées pour créer un CA pirate

MD5 Considered Harmful Today

Creating a rogue CA certificate

Alexander Sotirov
Marc Stevens
Jacob Appelbaum
Arjen Lenstra
David Molnar
Dag Arne Osvik
Benne de Weger

Creating an intermediate CA

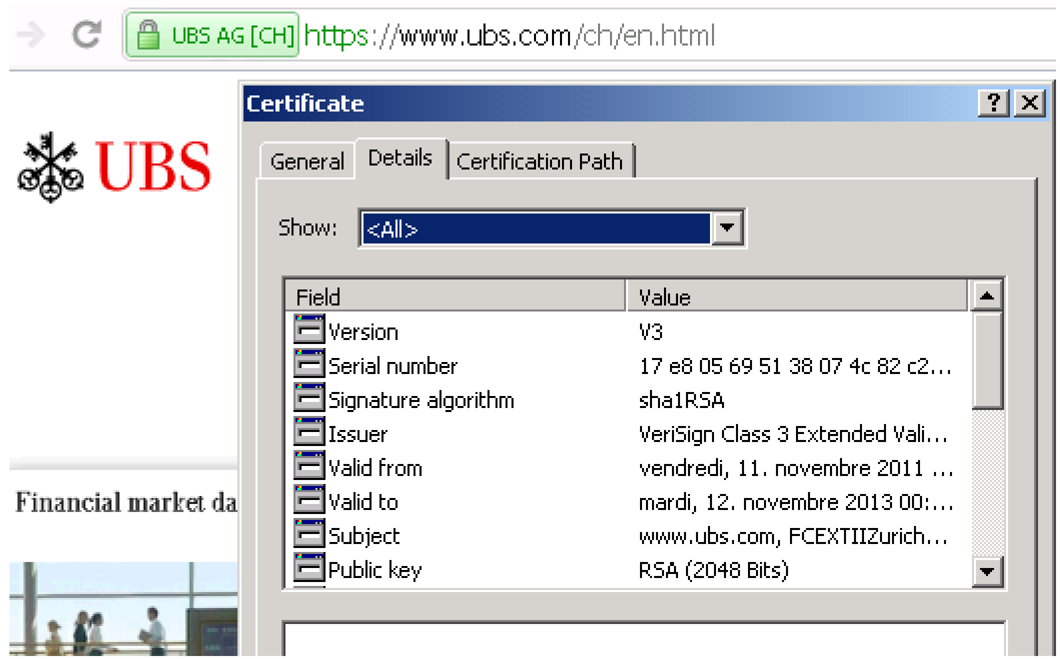
serial number	chosen prefix (difference)	rogue CA cert
validity period		
real cert domain name		rogue CA RSA key
		rogue CA X.509 extensions ← CA bit!
real cert RSA key	collision bits (computed)	Netscape Comment Extension (contents ignored by browsers)
X.509 extensions	identical bytes (copied from real cert)	
signature		signature

Chosen-prefix collisions

"For any targeted pair of distinct messages m_1 and m_2 we can effectively construct appendages b_1 and b_2 such that MD5($m_1 || b_1$) equals MD5($m_2 || b_2$)."

http://www.win.tue.nl/hashclash/TargetCollidingCertificates/

Et SHA-1?



February 15, 2005

SHA-1 Broken

SHA-1 has been broken. Not a reduced-round version. Not a simplified version. The real thing.

The research team of Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu (mostly from Shandong University in China) have been quietly circulating a paper describing their results:

- collisions in the the full SHA-1 in 2**69 hash operations, much less than the brute-force attack of 2**80 operations based on the hash length.

“new exact disturbance vector analysis for SHA-1 (…) we show how it can be used to implement both an identical-prefix and a **chosen-prefix collision attack on SHA-1**”

Marc Stevens, http://2012.sharcs.org/schedule.html

$$H(\text{ password })$$

pastebin.com/raw.php?i=e6H4TFen

```
````````````````````````\__\/``\:\/:/`\``\:\`\:\_\/`\__\/~
`````````````````````````__\::/```\``\:\`\:\`````````|``
`````````````````````````/``/:/``````\``\:_\/````````|``
````````````````````````/__/:/`````````\``\:\`````````|__
````````````````````````__\/```````````__\/``````````_
```

http://dhadingawaj.com.np/
username : admin
password : 3dd21daf392a550f13aa5c31d0b05fdb
username : rajan
password : 08a9fe930dcd0b55b860af8ce0e0936e

http://www.edupark.com.np/
username : admin
password : p3btredt02qN9TiL7PhCfg==
username : kunjan
password : vnLit/Zi12OG722L7PhCfg==
username : laxman
password : t3PvsuBw03ea92bqiMh1bQ==
username : sdsd
password : p3n/vZwAsgTpgAmY7PhCfg==

http://www.ird.gov.np
username : admin
password : b0b00f939f3d8a870b49e8ce02125215
username : manish
password : d3bb3afa8d6ef07ae9bc367ccf453edc

http://dntt.com.np/
username : DNTTTour
password : 052f3911bff8392414229b8ab79f4d9d

http://www.ecs.com.np/
username : ecsnepal
password : 1e2ebc9daf20fcef6a27567a5d1f6287
username : nabish

pastebin.com/raw.php?i=t1QGazKE

http://www.tinika-import.nl/ HACKED

---

Customer Names and Password Hashes:

fiwema

c9400c231b939fc28daf2870031f678d
cclaessen

bbfebe9d98336e4bd0c60345de1a061c
tereijntjes

7be4eb124fa31ab7ffe30e636eef354b
tereijntjes

15271129352ea83b009ddd8269de4d7b
ngfrankler

9fec61f2847417136e881c08d05c725d
medimitruk

f84b3106aa44544c19f299db28f9e779
feijgelsheim

57eb6a40b2be6ba471a890ba85e90b81
ahhfwisselo

c776505d9ee5ff78ed8ccd7d64edbdec
tereintjes

e2ec1ead2a674f7a15142f39abe4fd5c
kbultynck

841134c4b8a1ff2d91064df4fb8e9e16
```

H(password)

Requirements du système:

- ~~Mots de passe forts~~
- Si la DB des hash est connue, les passwords restent inconnus

H(password)

H requirements:

- Assez lent (dictionary attacks)
- Assez rapide (usabilité)
- Utiliser un “salt” (rainbow tables)

H(password)

H requirements:

- Assez lent (dictionary attacks)
- Assez rapide (usabilité)
- Utiliser un “salt” (rainbow tables)
- **“Non inversible”**

“Inverser” pour une DB de hashes =

- tous les pwds?
- 1 pwd particulier?
- 1 pwd (one-in-many attack)?
- 1 préimage (pas nécessairement le pwd)?

"Inverser" pour une DB de hashes =

- tous les pwds?
- 1 pwd particulier?
- 1 pwd (one-in-many attack)?
- 1 préimage (pas nécessairement le pwd)?

Un H peut-être resistant aux préimages (v1 et v2) mais facile à inverser pour retrouver des passwords!

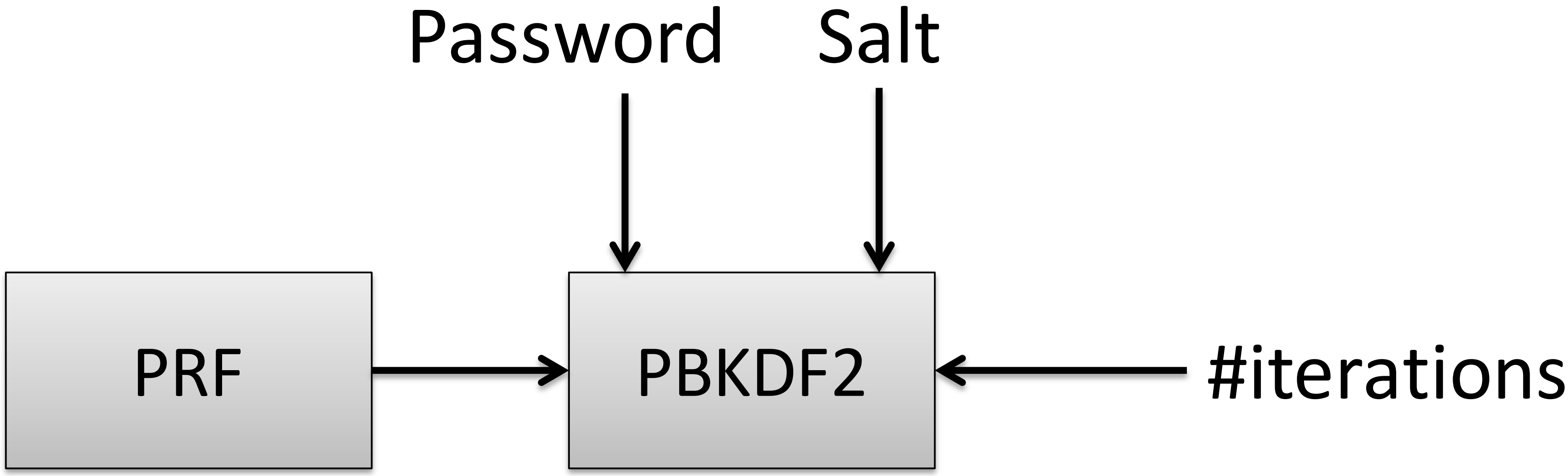
Standard: PBKDF2

Password-Based **Key Derivation** Function

PKCS#5 v2.0

RFC2898

WPA, iOS, BlackBerry, Truecrypt, WinZip, OpenOffice, etc.



PRF en général **HMAC-SHA-1**

Autres choix:

- HMAC-SHA-256, HMAC-SHA-3 (bientôt)
- MAC basé sur un block cipher (ex: CMAC, VMAC, Poly1305)

bcrypt (Provos/Mazières, USENIX 99)

Password-hash dédié basé sur le block cipher Blowfish

bcrypt (Provos/Mazières, USENIX 99)

Plus sûr que PBKDF2-HMAC-SHA-1 contre les GPUs (à #iterations équivalent)

PBKDF2-bcrypt?

SHA-3

Cryptographic Hash Project
Cryptographic Hash Algorithm Competition
Timeline for Hash Algorithm Competition
Federal Register Notices
Submission Requirements
ROUND 1
ROUND 2
ROUND 3 NEW!
Hash Forum
Contacts
Other Links

CSRC HOME > GROUPS > ST > HASH PROJECT

CRYPTOGRAPHIC HASH ALGORITHM COMPETITION

NIST announced a public competition (Federal Register Notice) on Nov. 2, 2007 to develop a new cryptographic hash algorithm, which converts a variable length message into a short "message digest" that can be used in generating digital signatures, message authentication codes, and many other security applications in the information infrastructure. The competition was NIST's response to advances in the cryptanalysis of hash algorithms. The winning algorithm will be named "SHA-3", and will augment the hash algorithms currently specified in the Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard.

NIST received sixty-four entries by October 31, 2008; and selected fifty-one candidate algorithms to advance to the first round on December 10, 2008, and fourteen to advance to the second round on July 24, 2009. A year was allocated for the public review of the fourteen second-round candidates.

NIST received significant feedback from the cryptographic community. Based on the public feedback and internal reviews of the second-round candidates,

Finding Collisions in the Full SHA-1

Xiaoyun Wang[1]*, Yiqun Lisa Yin[2], and Hongbo Yu[3]

[1] Shandong University, Jinan 250100, China, xywang@sdu.edu.cn
[2] Independent Security Consultant, Greenwich CT, US, yyin@princeton.edu
[3] Shandong University, Jinan250100, China, yhb@mail.sdu.edu.cn

Abstract. In this paper, we present new collision search attacks on the hash function SHA-1. We show that collisions of SHA-1 can be found with complexity less than 2^{69} hash operations. This is the first attack on the full 80-step SHA-1 with complexity less than the 2^{80} theoretical bound.

Keywords: Hash functions, collision search attacks, SHA-1, SHA-0.

CRYPTO 2005

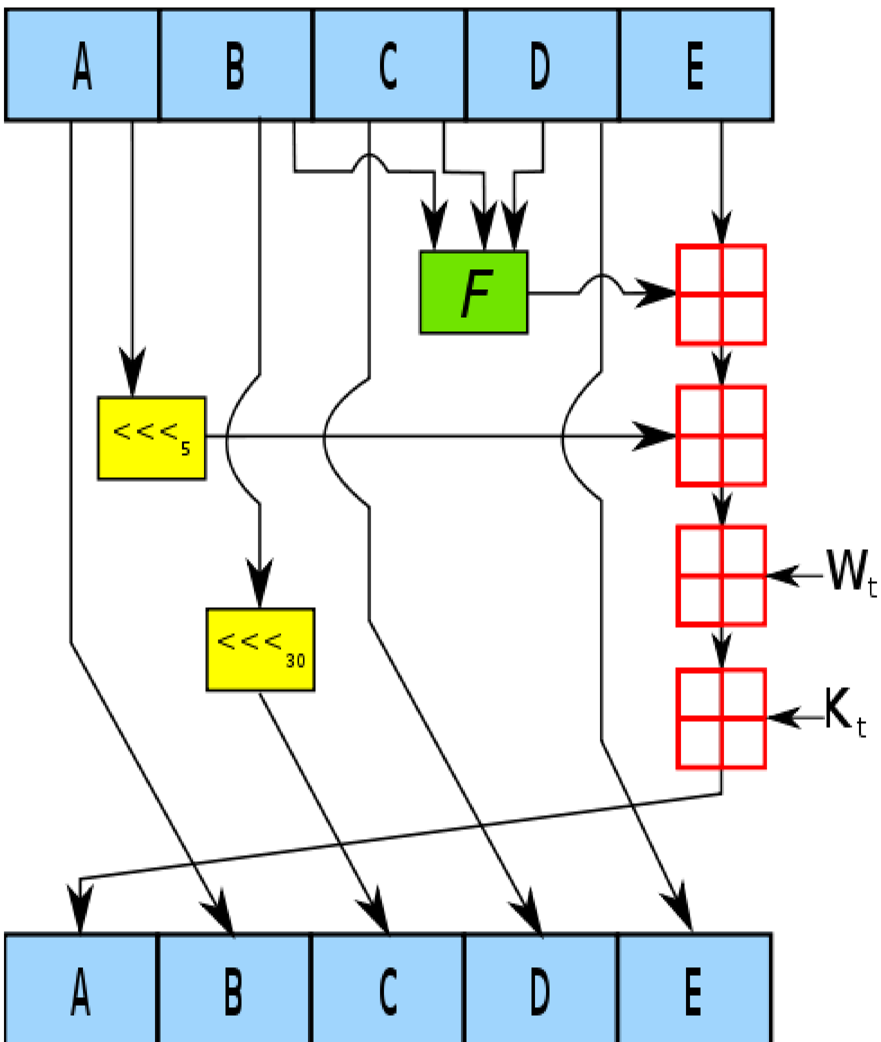
A Collision for 70-step SHA-1 in a Minute

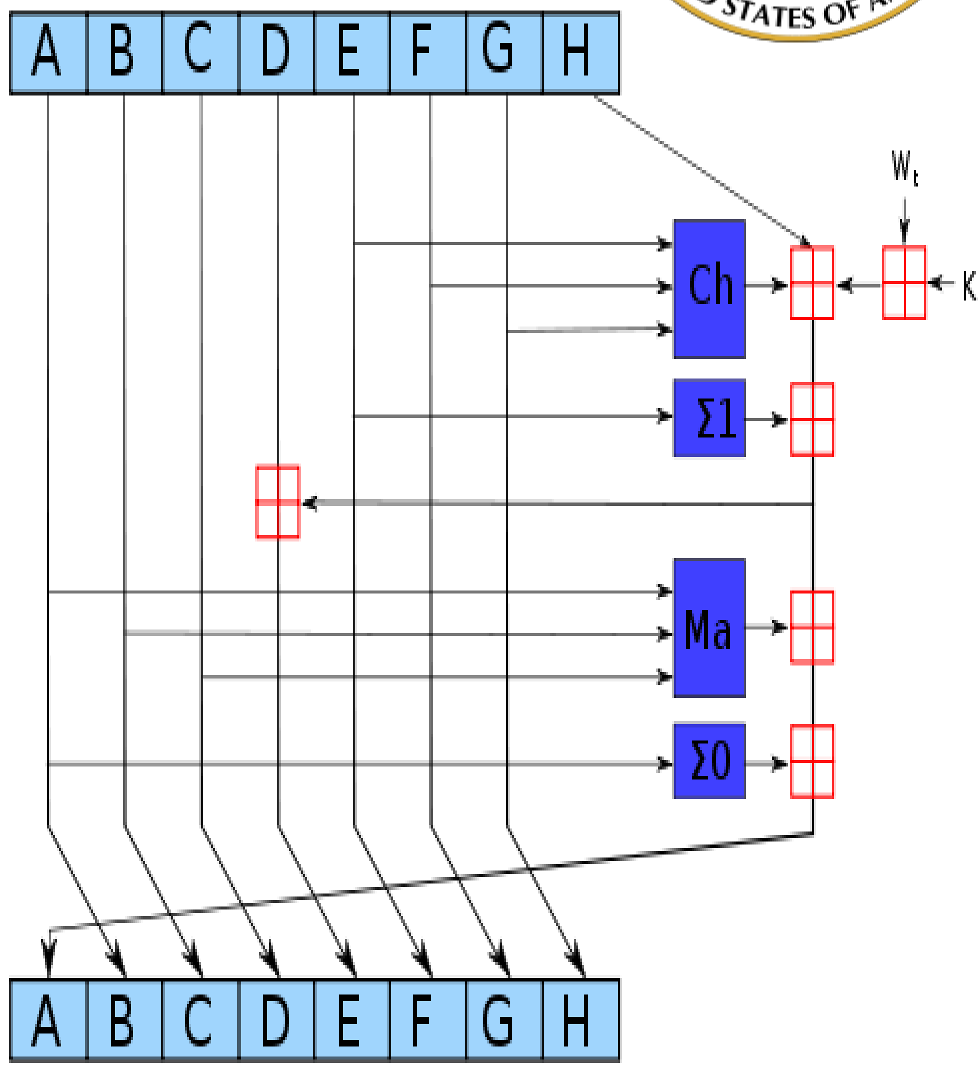
Christophe De Cannière and Florian Mendel
and Christian Rechberger

FSE 2007 Rump Session, 2007/03/27

Institute for Applied Information Processing and Communications (IAIK) - Krypto Group

Faculty of Computer Science
Graz University of Technology


A
B
C
D
E
F
<<<5
<<<30
Wt
Kt
A
B
C
D
E


A
B
C
D
E
F
G
H
Wt
Kt
Ch
Σ1
Ma
Σ0
A
B
C
D
E
F
G
H

2007

DEPARTMENT OF COMMERCE

National Institute of Standards and Technology

[Docket No.: 070911510–7512–01]

Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family

AGENCY: National Institute of Standards and Technology, Commerce.

ACTION: Notice and request for nominations for candidate hash algorithms.

SUMMARY: This notice solicits nominations from any interested party for candidate algorithms to be considered for SHA–3, and specifies how to submit a nomination package. It presents the nomination requirements and the minimum acceptability requirements of a "complete and proper" candidate algorithm submission. The evaluation criteria that will be used to appraise the candidate

4. Evaluation Criteria
  4.A Security
  4.B Cost
  4.C Algorithm and Implementation Characteristics
5. Initial Planning for the First SHA–3 Candidate Conference
6. Plans for the Candidate Evaluation Process
  6.A Overview
  6.B Round 1 Technical Evaluation
  6.C Round 2 Technical Evaluation
7. Miscellaneous

Authority: This work is being initiated pursuant to NIST's responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107–347.

1. Background

Modern, collision resistant hash functions were designed to create small, fixed size message digests so that a digest could act as a proxy for a possibly very large variable length message in a digital signature algorithm, such as RSA or DSA. These hash functions have since been widely used for many other "ancillary" applications, including hash-based message authentication codes, pseudo random number generators, and key derivation functions.

2007

that may be provided in a wide variety of cryptographic applications, including digital signatures (FIPS 186–2), key derivation (NIST Special Publication 800–56A), hash-based message authentication codes (FIPS 198), deterministic random bit generators (SP 800–90), and additional applications that may be brought up by NIST or by the public during the evaluation process. Claimed applications of the hash functions will be evaluated for their practical importance if this evaluation is necessary for comparing the submitted hash algorithms.

ii. Specific Requirements When Hash Functions Are Used To Support HMAC, Pseudo Random Functions (PRFs), and Randomized Hashing

algorithm constructions; any result that shows that the candidate algorithm does not meet these requirements will be considered to be a serious attack.

- Collision resistance of approximately $n/2$ bits,
- Preimage resistance of approximately n bits,
- Second-preimage resistance of approximately n-k bits for any message shorter than 2^k bits,
- Resistance to length-extension attacks, and
- Any m-bit hash function specified by taking a fixed subset of the candidate function's output bits is expected to meet the above requirements with m replacing n. (Note that an attacker can choose the m-bit subset specifically to allow a limited number of precomputed

2007

their resistance against attacks discovered during the evaluation process, and for their likelihood of resistance against future attacks.

v. Other Consideration Factors

In addition to the evaluation factors mentioned above, the quality of the security arguments/proofs, the clarity of the documentation of the algorithm, the quality of the analysis on the algorithm performed by the submitters, the simplicity of the algorithm, and the confidence of NIST and the cryptographic community in the algorithm's long-term security may all be considered.

2008

64 soumissions

SONY

INRIA

LETTRES SCIENCES

ÉCOLE NORMALE SUPÉRIEURE

certicom™

Technion
Israel Institute of Technology

Sandia National Laboratories

gemalto

Microsoft®

2008

64 soumissions

BT

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

KOREA UNIVERSITY 1905

KATHOLIEKE UNIVERSITEIT LEUVEN

DTU

n|w Fachhochschule Nordwestschweiz

NXP

Loughborough University

2008

64 soumissions

IBM

EPFL
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

EADS

QUALCOMM®

ETH

HITACHI
Inspire the Next

intel®

TU Graz

orange™

2009

14 “second-round candidates“

BLAKE	Blue Midnight Wish	CubeHash	ECHO
Fugue	Grøstl	Hamsi	JH
Keccak	Luffa	Shabal	SHAvite-3
SIMD	Skein		

2010

5 finalistes

BLAKE

Grøstl

JH

Keccak

Skein

2011

Derniers tweaks aux algorithmes

Plus de cryptanalyse

Plus d’implémentations hardware

Plus d’implémentations software

2012

22-23 mars : final SHA-3 conference

Q2/été: sélection de SHA-3

Secure?

ON MY WAY
57
31

Rotational Rebound Attacks on Reduced Skein

Dmitry Khovratovich[1,2], Ivica Nikolić[1], and Christian Rechberger[3]

[1]: University of Luxembourg; [2]: Microsoft Research Redmond, USA;
[3]: Katholieke Universiteit Leuven, ESAT/COSIC, and IBBT, Belgium
dkhovrat@microsoft.com, ivica.nikolic@uni.lu,
christian.rechberger@esat.kuleuven.be

How to Improve Rebound Attacks*

María Naya-Plasencia†

FHNW, Windisch, Switzerland

~ 45 articles de cryptanalyse rien que pour les 5 finalistes

Collisions for variants of the BLAKE hash function

Janoš Vidali[a], Peter Nose[a], Enes Pašalić[b]

[a] *University of Ljubljana, FRI, Ljubljana, Slovenia,*
e-mail: {janos.vidali, peter.nose}@fri.uni-lj.si
[b] *University of Primorska, FAMNIT, Koper, Slovenia, e-mail: enespasalic@yahoo.se*

Unaligned Rebound Attack: Application to Keccak

andre Duc[1], Jian Guo[2], Thomas Peyrin[3], and Lei Wei[3]

Ecole Polytechnique Fédérale de Lausanne, Switzerland
[2] Institute for Infocomm Research, Singapore
[3] Nanyang Technological University, Singapore
alexandre.duc@epfl.ch
{ntu.guo,thomas.peyrin}@gmail.com
wl@pmail.ntu.edu.sg

Performance?

$ gcc sha3_x64.c –O3 –march=native –funroll-loops

14.25 cycles/byte

$ gcc sha3_x64.c –O3 –march=native –funroll-loops

14.25 cycles/byte

$ gcc sha3_sse2.c –O2 –fomit-frame-pointer –fprefetch-loop-arrays

15.42 cycles/byte

$ gcc sha3_x64.c –O3 –march=native –funroll-loops

14.25 cycles/byte

$ gcc sha3_sse2.c –O2 –fomit-frame-pointer –fprefetch-loop-arrays

15.42 cycles/byte

$ gcc sha3_sse2.c –O3 –m64 -march=corei7-avx

13.98 cycles/bytes

$ gcc sha3_x64.c –O3 –march=native –funroll-loops

14.25 cycles/byte

$ gcc sha3_sse2.c –O2 –fomit-frame-pointer –fprefetch-loop-arrays

15.42 cycles/byte

$ gcc sha3_sse2.c –O3 –m64 -march=corei7-avx

13.98 cycles/bytes

$ gcc sha3_ssse3.c –O3 –m64 -march=corei7-avx –fomit-frame-pointer

13.44 cycles/bytes

VAMPIRE

eBACS: ECRYPT Benchmarking of Cryptographic Systems

ECRYPT II

General information:	Introduction	eBASH	eBASC	eBATS	**SUPERCOP**	XBX	Computers
How to submit new software:	Hash functions		Stream ciphers	DH functions	Public-key encryption	Public-key signatures	
List of primitives measured:	SHA-3 candidates	All hash functions	Stream ciphers	DH functions	Public-key encryption	Public-key signatures	
Measurements indexed by machine:	SHA-3 candidates	All hash functions	Stream ciphers	DH functions	Public-key encryption	Public-key signatures	

SUPERCOP

SUPERCOP is a toolkit developed by the VAMPIRE lab for measuring the performance of cryptographic software. SUPERCOP stands for **S**ystem for **U**nified **P**erformance **E**valuation **R**elated to **C**ryptographic **O**perations and **P**rimitives; the name was suggested by Paul Bakker.

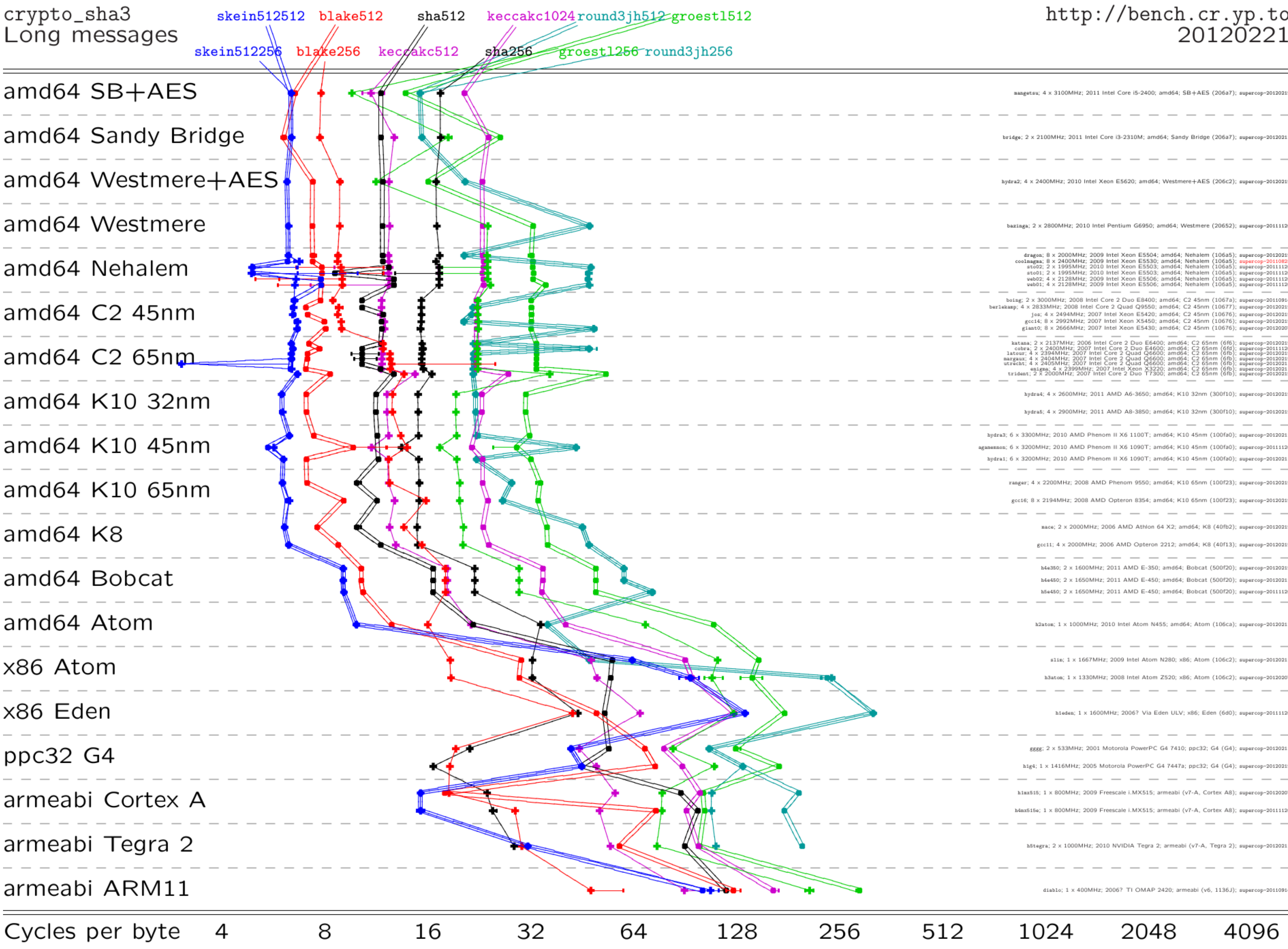
The latest release of SUPERCOP measures the performance of hash functions, secret-key stream ciphers, public-key encryption systems, public-key signature systems, and public-key secret-sharing systems. SUPERCOP integrates and improves upon

- STVL's benchmarking suite for stream ciphers submitted to eSTREAM, the ECRYPT Stream Cipher Project (which finished in April 2008);
- VAMPIRE's BATMAN (**B**enchmarking of **A**symmetric **T**ools on **M**ultiple **A**rchitectures, **N**on-interactively) suite for public-key systems submitted to the eBATS (**E**CRYPT **B**enchmarking of **A**symmetric **S**ystems) project; and
- additional tools developed for VAMPIRE's new eBASH (**E**CRYPT **B**enchmarking of **A**ll **S**ubmitted **H**ashes) project.

Specifically, SUPERCOP measures cryptographic primitives according to several criteria:

“SUPERCOP automatically tries all available implementations of each primitive, and many compilers for each implementation, to select the fastest combination of implementation and compiler. “

http://bench.cr.yp.to/supercop.html


crypto_sha3
Long messages
http://bench.cr.yp.to
20120221
skein512512
blake512
sha512
keccakc1024
round3jh512
groestl512
skein512256
blake256
keccakc512
sha256
groestl256
round3jh256
amd64 SB+AES
amd64 Sandy Bridge
amd64 Westmere+AES
amd64 Westmere
amd64 Nehalem
amd64 C2 45nm
amd64 C2 65nm
amd64 K10 32nm
amd64 K10 45nm
amd64 K10 65nm
amd64 K8
amd64 Bobcat
amd64 Atom
x86 Atom
x86 Eden
ppc32 G4
armeabi Cortex A
armeabi Tegra 2
armeabi ARM11
Cycles per byte
4
8
16
32
64
128
256
512
1024
2048
4096

SHA-3 proposal BLAKE

Final BLAKE   Downloads   Cryptanalysis   Software implementations   Hardware implementations

BLAKE is one of the five hash functions in the final of the NIST SHA-3 Competition. BLAKE is one of the simplest designs to implement, and it entirely relies on previously analyzed components: the HAIFA structure and the ChaCha core function.

The two main instances of BLAKE are BLAKE-256 and BLAKE-512. They respectively work with 32- and 64-bit words, and produce 256- and 512-bit digests.

http://131002.net/blake/

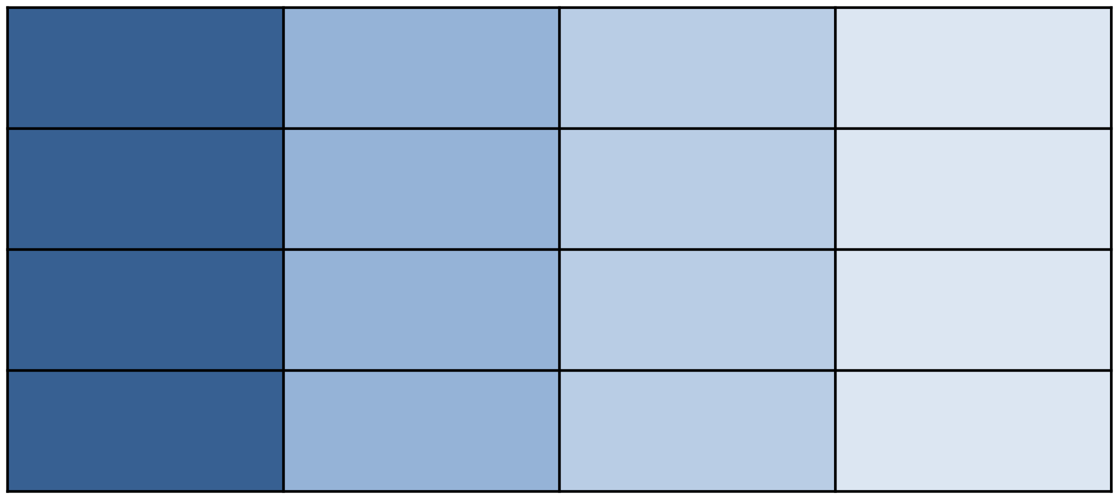
Jean-Philippe Aumasson (FHNW, NAGRA)

Luca Henzen (ETHZ, UBS)

Willi Meier (FHNW)

Raphael C.-W. Phan (Loughborough Uni, UK)

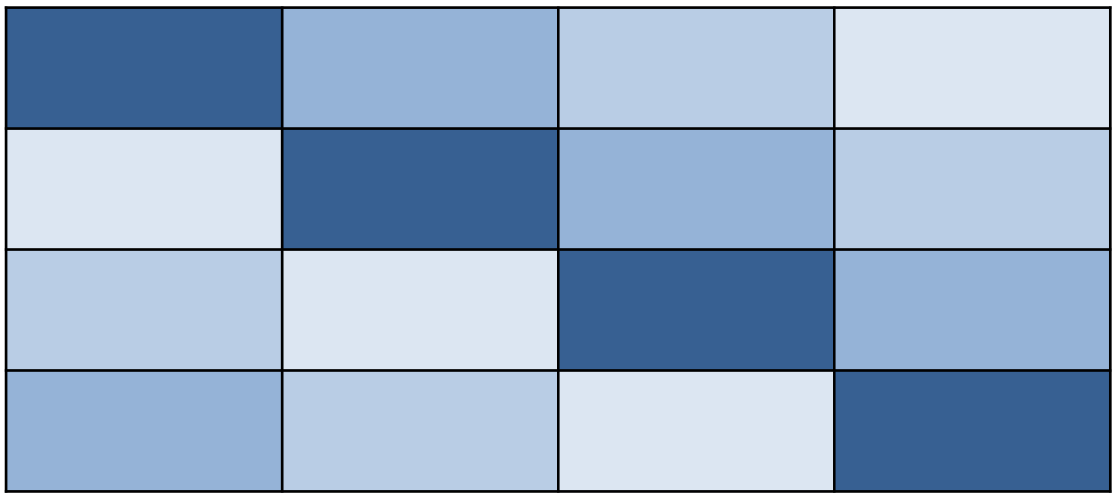
BLAKE core = permutation d'un état 4x4



Mots de 32 bits pour BLAKE-256

Mots de 64 bits pour BLAKE-512

BLAKE core = permutation d’un état 4x4



Mots de 32 bits pour BLAKE-256

Mots de 64 bits pour BLAKE-512

La transformation **G**

$a \mathrel{+}= X \oplus const$

$a \mathrel{+}= b$

$d = (d \oplus a) \ggg 16$

$c \mathrel{+}= d$

$b = (b \oplus c) \ggg 12$

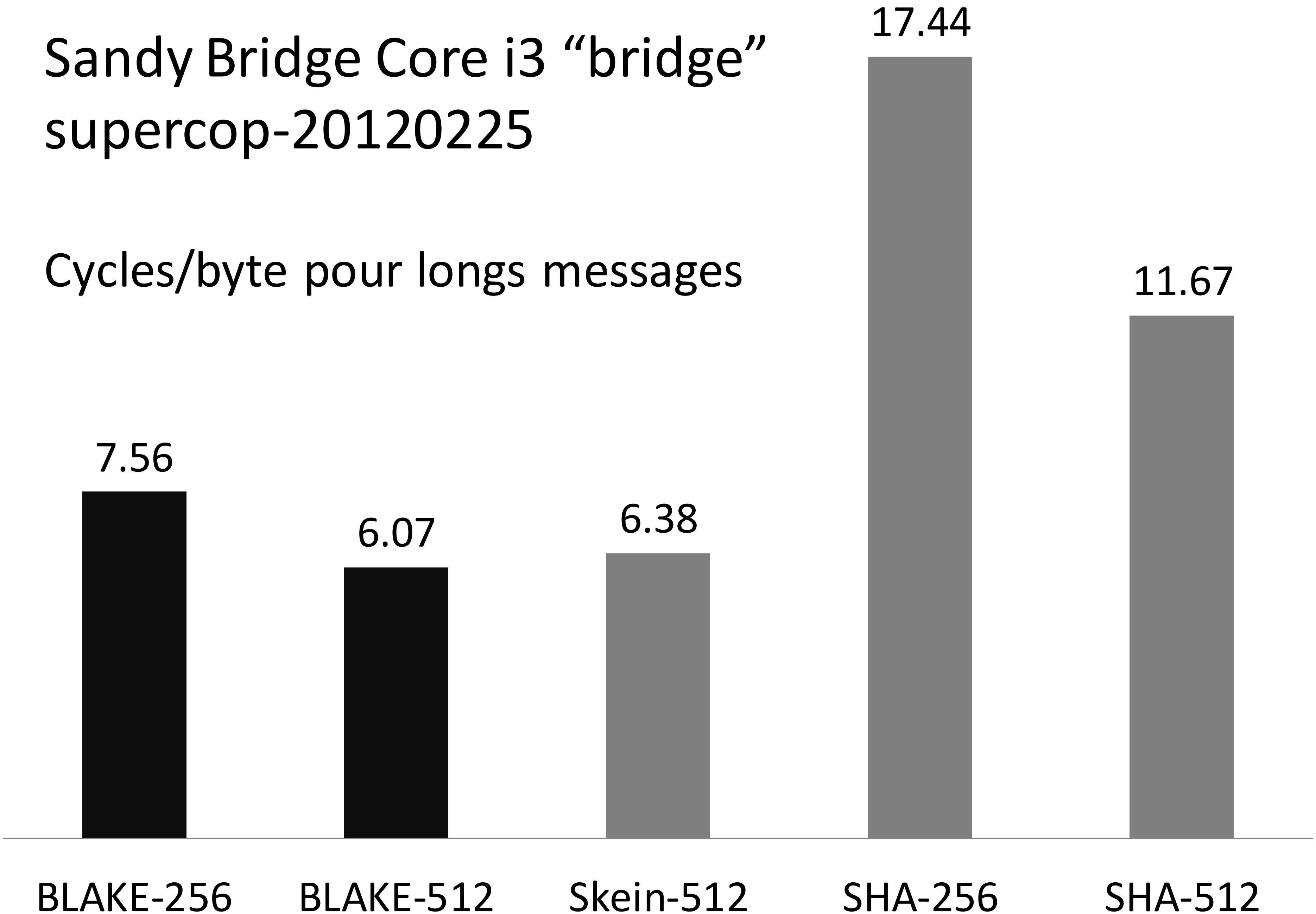
$a \mathrel{+}= Y \oplus const$

$a \mathrel{+}= b$

$d = (d \oplus a) \ggg 8$

$c \mathrel{+}= d$

$b = (b \oplus c) \ggg 7$


Sandy Bridge Core i3 “bridge”
supercop-20120225
Cycles/byte pour longs messages
17.44
11.67
7.56
6.07
6.38
BLAKE-256
BLAKE-512
Skein-512
SHA-256
SHA-512

Conclusion

On peut (presque) tout faire avec les fonction de hash

SHA-1 est vieillissant.

Mais SHA-3 arrive bientôt!

Merci!

Insomni'Hash

Jean-Philippe Aumasson