Cryptanalysis of Stream-Based Hashes

ECRYPT II
Hash^3: Proofs, Analysis, and Implementation

Thomas Peyrin
Ingenico

November 17th 2009 - Tenerife

Outline

Stream-based hash functions

- What is a stream-based hash function ?
- Some examples
- A perfect example: CubeHash

Generic attacks

- Meet-in-the-middle attacks
- Slide attacks

Differential attacks

- Linear differential paths
- Truncated differential paths
- Symmetric differential paths

Using the freedom degrees

Outline

Outline

Stream-based hash functions

- What is a stream-based hash function ?
- Some examples
- A perfect example: CubeHash

Generic attacks

- Meet-in-the-middle attacks
- Slide attacks

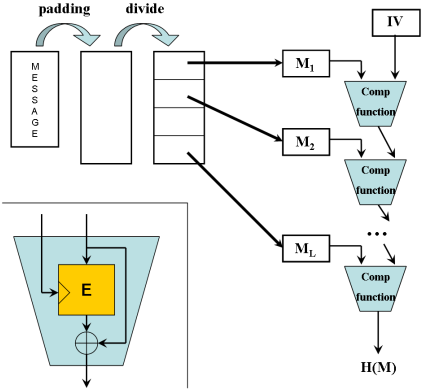
Differential attacks

- Linear differential paths
- Truncated differential paths
- Symmetric differential paths

Using the freedom degrees

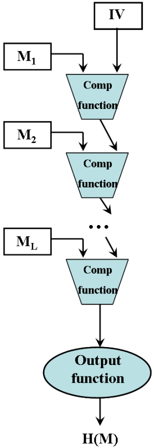
How to build a hash function (usually) ?

Merkle-Damgård algorithm + Davies-Meyer



How to build a hash function (usually) ?

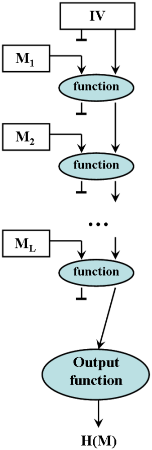
Generalization



- use an output function (for example truncation in double pipe construction)
- use another method for introducing message chunks
- **c** represents the capacity.
- **r** represents the bit-rate.
- **R** represents the number of rounds.

How to build a hash function (usually) ?

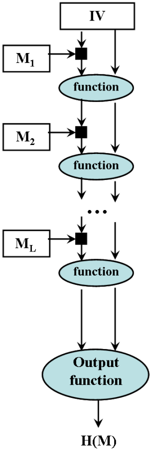
Generalization



- use an output function (for example truncation in double pipe construction)
- use another method for introducing message chunks
- **c** represents the capacity.
- **r** represents the bit-rate.
- **R** represents the number of rounds.

How to build a hash function (usually) ?

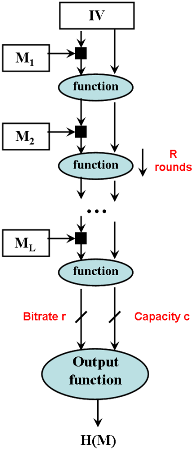
Generalization



- use an output function (for example truncation in double pipe construction)
- use another method for introducing message chunks
- **c** represents the capacity.
- **r** represents the bit-rate.
- **R** represents the number of rounds.

How to build a hash function (usually) ?

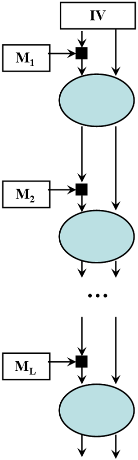
Generalization



- use an output function (for example truncation in double pipe construction)
- use another method for introducing message chunks
- **c** represents the capacity.
- **r** represents the bit-rate.
- **R** represents the number of rounds.

What is a stream-based hash function ?

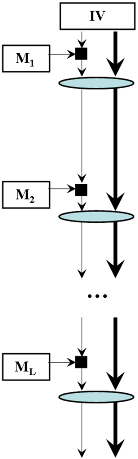
Block/stream-based hash functions



- **block-based hash:**
  r is bigger or at least the same size than c
- **stream-based hash:**
  r is small compared to c
- increasing c/r improves security: less control to the attacker
- increasing R improves security: less good differential paths
- stream-based hashes internal function is in general a permutation

What is a stream-based hash function ?

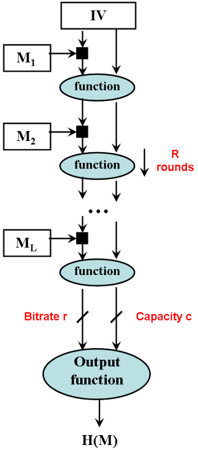
Block/stream-based hash functions



- **block-based hash:**
  r is bigger or at least the same size than c
- **stream-based hash:**
  r is small compared to c
- increasing c/r improves security: less control to the attacker
- increasing R improves security: less good differential paths
- stream-based hashes internal function is in general a permutation

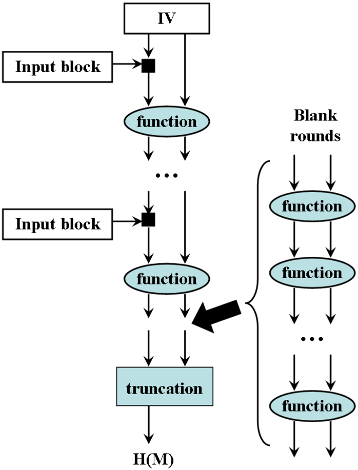
What is a stream-based hash function ?

Block/stream-based hash functions



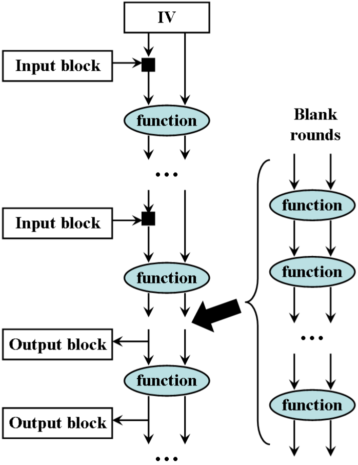
- **block-based hash:**
  r is bigger or at least the same size than c
- **stream-based hash:**
  r is small compared to c
- increasing c/r improves security: less control to the attacker
- increasing R improves security: less good differential paths
- stream-based hashes internal function is in general a permutation

Output function



- you can add **blank rounds** (Grindahl, RadioGatun, Lux, CubeHash, ...)
- **option 1:** just truncate the internal state to obtain the hash value (Grindahl, CubeHash, ...)
- **option 2:** you can continue iterating the round function without introducing messages and slowly extracting chunk of the internal state to build the hash value (RadioGatun, Lux, Keccak, ...)

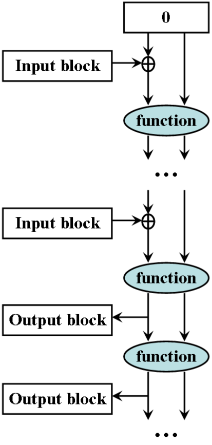
Output function



- you can add **blank rounds** (Grindahl, RadioGatun, Lux, CubeHash, ...)
- **option 1:** just truncate the internal state to obtain the hash value (Grindahl, CubeHash, ...)
- **option 2:** you can continue iterating the round function without introducing messages and slowly extracting chunk of the internal state to build the hash value (RadioGatun, Lux, Keccak, ...)

Outline

The sponge functions [BDPV-ECRYPTHW07]



- sponge functions: introduced by Bertoni, Daemen, Peeters and Van Assche in 2007.
- Example: Keccak.
- **Particularities:**
  - special padding rule (that implies last message block $\neq 0$)
  - insert message chunks with a XOR
  - use squeezing process as output function (with the same internal state words than insertion): allows to be very flexible on the hash output size, with the same internal function ... can be used as a stream cipher.
  - the round function should presents no structural property (hermetic sponge)

Security proofs for sponge functions [BDPV-EC08]

- **white box model**: the attacker has access to the internal round function. Use the indifferentiability framework from Maurer *et al.* [MRH-TCC04].
- assume the internal function is a random permutation
- **Theorem:** a random sponge can be differentiated from a random oracle only with probability $\simeq N(N+1)/2^{c+1}$, with $N < 2^c$, where N is the total number of calls to the internal round function.
- generic attacks require $2^{c/2}$.
- as long as you can't say anything on the actual internal permutation (i.e. structural properties), the sponge looks like a random oracle (resistant to multicollisions, long 2nd preimage, length-extension attacks, ...)

Examples

name	capacity c	bit-rate r	nb rounds R	insert function	output function
Panama	8480	256	1	XOR	BR + trunc
RadioGatun	1760	96	1	XOR	BR + squeeze
Keccak	512	1088	24	XOR	squeeze
Grindahl	384	32	1	ERASE	BR + trunc
Fugue	928	32	1	XOR	BR + trunc
LUX	736	32	1	XOR	BR + squeeze
HAMSI	256	32	3	special	BR + trunc
Luffa	512	256	1	special	BR + squeeze
CubeHash	768	256	16	XOR	BR + trunc
EnRupt	576	64	8	XOR	BR + squeeze
SHABAL	896	512	3	MIX	BR + squeeze

Outline

Stream-based hash functions

- What is a stream-based hash function ?
- Some examples
- A perfect example: CubeHash

Generic attacks

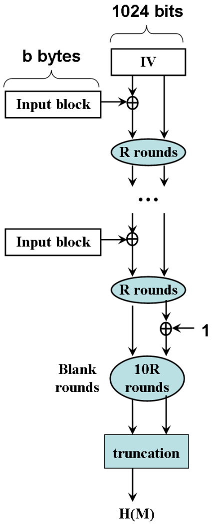
- Meet-in-the-middle attacks
- Slide attacks

Differential attacks

- Linear differential paths
- Truncated differential paths
- Symmetric differential paths

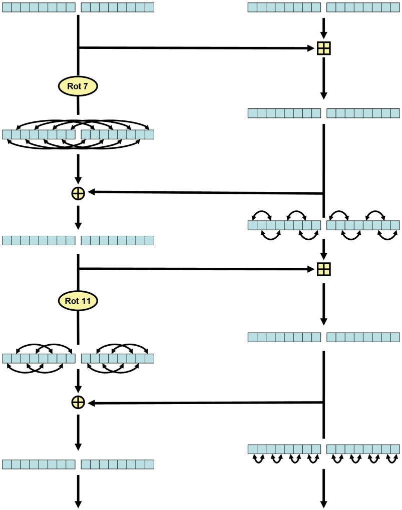
Using the freedom degrees

CubeHash-R/b



- SHA-3 candidate of Dan Bernstein
- internal state of 1024 bits (32 words of 32 bits each)
- insert b bytes of message (with xor) each iteration
- process R rounds of the permutation each iteration
- xor 1 and execute 10R blank rounds
- truncate the internal state to the appropriate hash size
- capacity = 1024 - 8b.

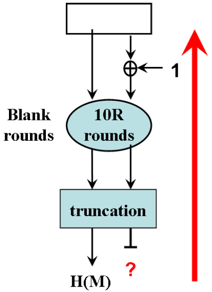
CubeHash round function



Outline

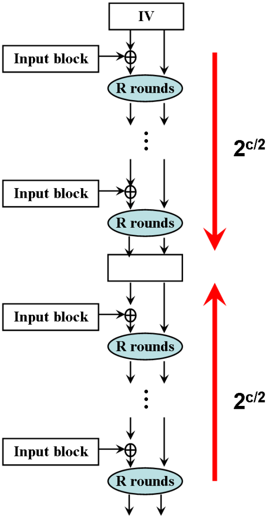
Outline

Meet-in-the-middle attacks



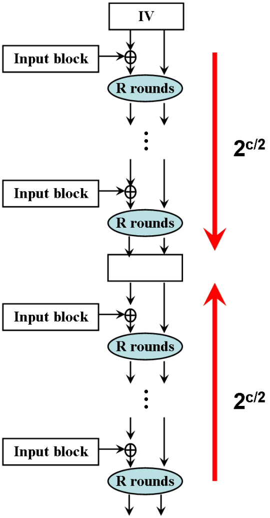
- assume we use an internal permutation, let's try to find a preimage
- invert the output function
- compute $2^{c/2}$ candidates forward and backward ...
- ... and meet-in-the-middle
- to be preimage resistant, the capacity should be $c \geq 2n$
- in the case of CubeHash, we need $2^{(1024-8b)/2} = 2^{512-4b}$ operations to find a preimage

Meet-in-the-middle attacks



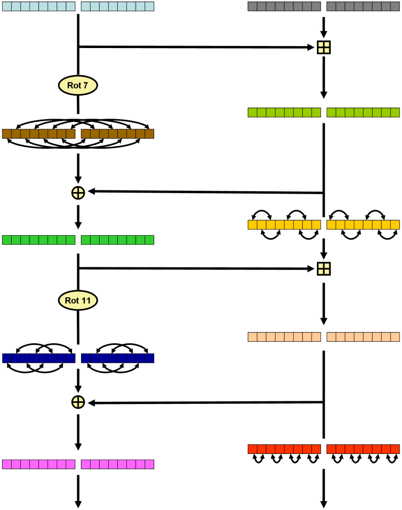
- assume we use an internal permutation, let's try to find a preimage
- invert the output function
- compute $2^{c/2}$ candidates forward and backward ...
- ... and meet-in-the-middle
- to be preimage resistant, the capacity should be $c \geq 2n$
- in the case of CubeHash, we need $2^{(1024-8b)/2} = 2^{512-4b}$ operations to find a preimage

Meet-in-the-middle attacks



- assume we use an internal permutation, let's try to find a preimage
- invert the output function
- compute $2^{c/2}$ candidates forward and backward ...
- ... and meet-in-the-middle
- to be preimage resistant, the capacity should be $c \geq 2n$
- in the case of CubeHash, we need $2^{(1024-8b)/2} = 2^{512-4b}$ operations to find a preimage

CubeHash round function structural property [ABMNP-ACISP09]



Improving meet-in-the-middle attacks with structural property

- find all symmetry classes in CubeHash internal function
- 16 classes of 2^{512} elements each, a total of 2^{516} symmetric states
- **let's find a preimage:**
    - invert the output function
    - from this internal state, compute backward and reach a symmetric state ($2^{1024-516-8b} = 2^{508-8b}$ operations)
    - from the IV, compute forward and reach a symmetric state ($2^{1024-516-8b} = 2^{508-8b}$ operations)
    - do a meet-in-the-middle while remaining in the symmetry class $S_F \cap S_B$ ($2^{512/2} = 2^{256}$ operations)
- total complexity $2^{512-8b} < 2^{512-4b}$

Outline

Stream-based hash functions
- What is a stream-based hash function ?
- Some examples
- A perfect example: CubeHash

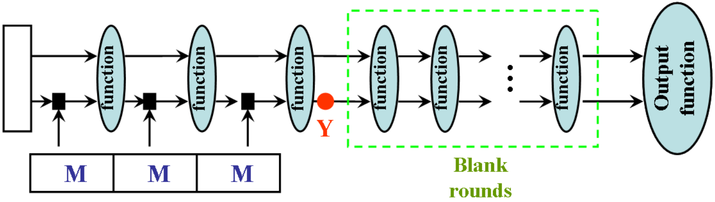
Generic attacks
- Meet-in-the-middle attacks
- Slide attacks

Differential attacks
- Linear differential paths
- Truncated differential paths
- Symmetric differential paths

Using the freedom degrees

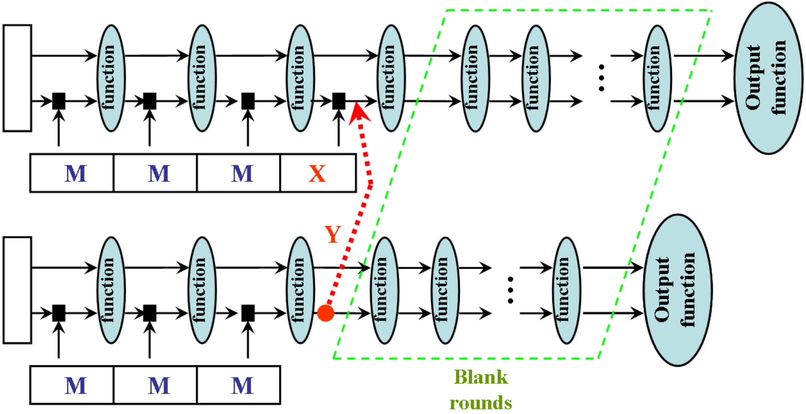
Slide attacks on stream-based hash functions

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



Slide attacks on stream-based hash functions

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



Slide attacks for hash functions

What can we obtain from slide attacks ?

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

We'll try to attack the following MAC construction:

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

Slide attacks for hash functions

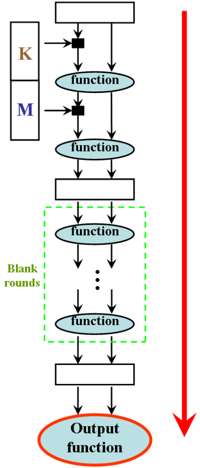
We'll try to attack the following MAC construction:

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

- **... which is secure if the hash function is modeled as a random oracle.**
- **Merkle-Damgård already known to be weak against this construction**: given $\mathrm{MAC}(K, M) = H(K||M)$, compute $\mathrm{MAC}(K, M||Y) = H(K||M||Y)$ without knowing the secret key K.
- patch provided in Coron *et al.*'s paper [CDMP-CRYPTO05].

Slide attacks on stream-based hash functions

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



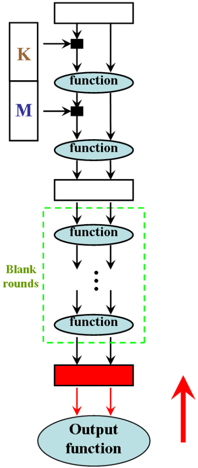
The attack will be in three steps:

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

The padding must also be taken in account !

Slide attacks on stream-based hash functions

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



The attack will be in three steps:

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

The padding must also be taken in account !

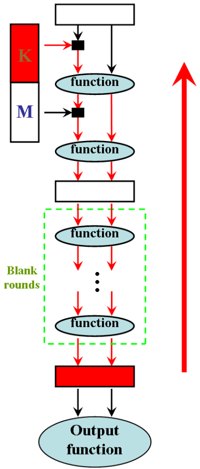
Slide attacks on stream-based hash functions

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

The attack will be in three steps:

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

The padding must also be taken in account !



Find and detect slid pairs of messages.

If you are inserting message blocks with XOR:

- very easy to slide, just use 0
- you don't need to detect it, you know it will slide
- impossible in the original sponge framework (in which the last inserted word must be different from 0) ...
- ... but possible if a different padding is used !

If you are inserting message blocks with ERASE:

- you can slide if you replace exactly what you erased
- happens with probability $P = 2^{-r}$
- detection depends on the output function:
  - very easy with the squeezing process (all the output words are shifted by one iteration).
  - more complicated with a direct truncation.

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

Patches

This attacks works against:

- Grindahl [GLP-AC08]
- LUX [P-SHA3list09] (with chosen salt)

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

- add a constant to the internal state just before the blank rounds to clearly separate them from the normal rounds (CubeHash).
- use a different transformation during the blank rounds (Panama, SHABAL).
- **If you're inserting message blocks with a XOR:** just use exactly the sponge framework and make sure that the last inserted message work is different from zero (RadioGatun, Keccak, EnRupt).

Outline

General principles

- **internal collisions:** the collision occurs on the whole internal state, before the blank rounds or the output function.
  - **advantages:** you can use the freedom degrees MUCH more efficiently
  - **drawbacks:** you have to collide on the entire big internal state
- **external collisions:** the internal state before the blank rounds or the output function contains some differences:
  - **advantages:** you only have to collide on the (small) hash output size.
  - **drawbacks:** you have no freedom degree to use
- Finding internal near collisions is useless (the output function is often strong because it doesn't affect the efficiency for long messages)
- Finding free-start collisions is useless in practice (and very easy since one can invert the internal function)

General principles

Situation different for stream-cipher based and block-based hash functions (not a rule, just a general observation):

- **block-based:**
    - finding a differential path is not the most difficult part (e.g. we know very good differential paths for SHA-1)
    - using the freedom degrees is hard, because they are all located at the same place while the conditions are everywhere (many freedom degrees are wasted in SHA-1 attacks) you may find good differential characteristics for the internal functions used in stream-based hashes ...
    - ... but the problem is how to link them
- **stream-based:**
    - finding a differential path is hard, because the internal state is really big (many conditions to take care of). Moreover, you often need several iterations in order to get a collision.
    - using the freedom degrees is rather easy, because you only have a few incoming each iteration: it is easy to use each of them to take care of a condition.

Outline

Stream-based hash functions
- What is a stream-based hash function ?
- Some examples
- A perfect example: CubeHash

Generic attacks
- Meet-in-the-middle attacks
- Slide attacks

Differential attacks
- Linear differential paths
- Truncated differential paths
- Symmetric differential paths

Using the freedom degrees

Linear differential paths

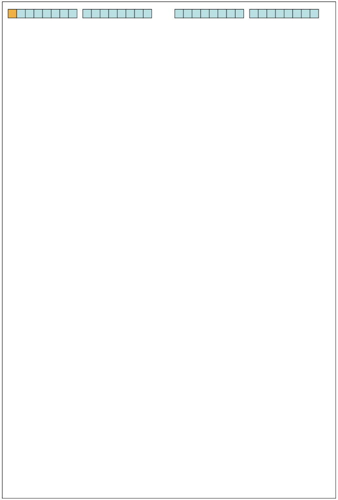
- try to **linearize** the scheme ... for example, simply replace additions by XORs
- solve the set of linear equations and try to find a good differential trail (in general, good = low weight)

Complexity computation:

- **two situations** have to be considered in order to compute the success probability of the differential path in the non-linearized case (both with probability 1/2):
  - **move:** a perturbation at a certain bit position is added to another bit containing no difference.
  - **correction:** a perturbation at a certain bit position is added to another bit containing a difference.
- for the addition of two words $A + B$, the probability of a linear behavior is $\mathrm{HW}((\Delta_A \vee \Delta_B) \wedge \mathtt{0x7fffffff})$.

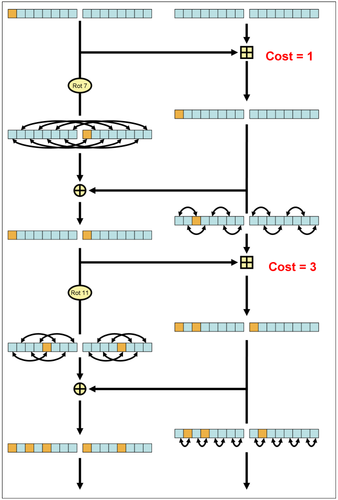
Works rather well for (32 or 64-bit)-word oriented primitives (ex: EnRupt [IP-FSE09], CubeHash [D-SHA3list09] [BP-ACNS09] [BKMP-AC09])

Linear differential paths: example for CubeHash-2/4



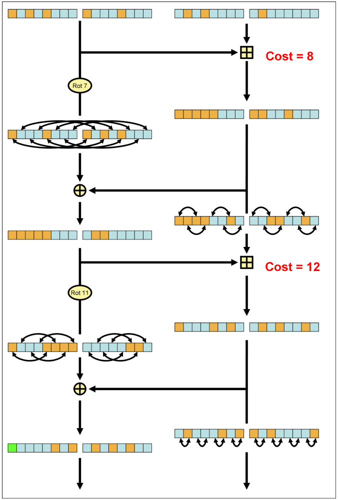
- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Linear differential paths: example for CubeHash-2/4



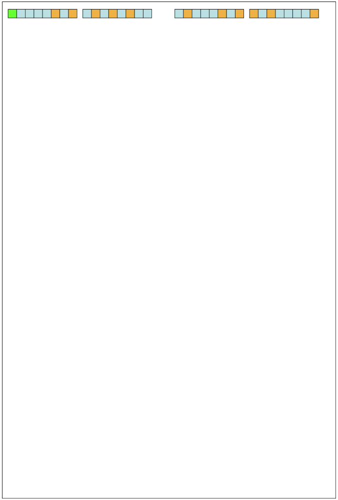
- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Linear differential paths: example for CubeHash-2/4



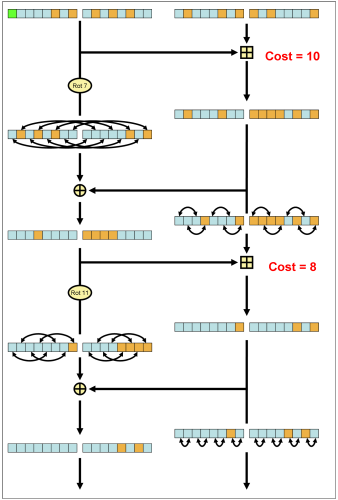
- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Linear differential paths: example for CubeHash-2/4



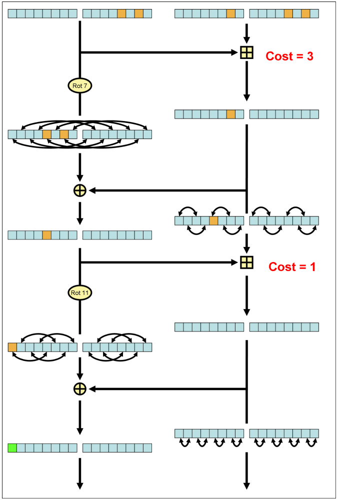
- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Linear differential paths: example for CubeHash-2/4



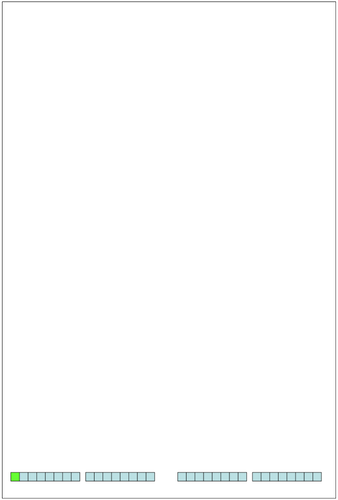
- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Linear differential paths: example for CubeHash-2/4



- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

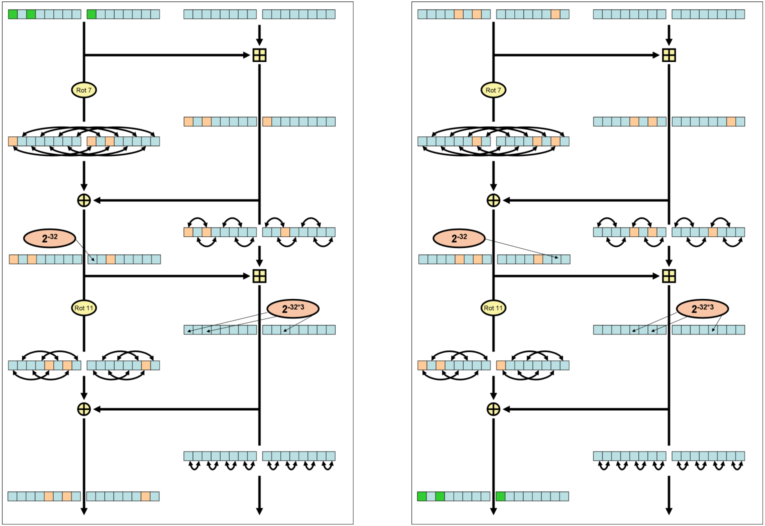
Linear differential paths: example for CubeHash-2/4



- add a one bit difference on X_0 (at position i).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at positions i+4, i+14, i+22).
- do one iteration (2 rounds).
- erase all the differences in X_0 (at position i+4).
- 46 bit conditions in total.

Outline

Truncated differential paths: CubeHash-1/36 example



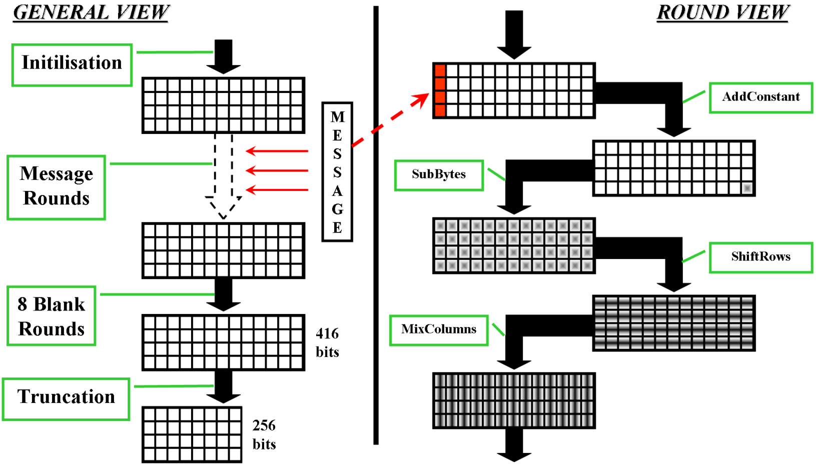
Truncated differential paths: CubeHash-1/36 results

Results (using freedom degrees):

- a collision for CubeHash-1/36 in 2^{32} operations.
- a collision for CubeHash-2/36 in 2^{96} operations.
- ... seems hard to go further !

Truncated differential paths don’t work well for CubeHash.

Truncated differential paths: Grindahl



Truncated differential paths: attacking Grindahl [P-AC07]

- Building a differential path is really hard because of the two security properties:
  - a collision requires intermediate states with at least half of the bytes active.
  - an internal collision requires at least 5 rounds.
- **idea - take the all-difference state as a check point:**
  - from a no-difference state to an all-difference state: hopefully very easy ! No need for a differential path here.
  - from an all-difference state to a no-difference state: harder ! Build the differential path backward and search for a collision onward.
- the costly part when searching for a collision is obviously the second stage !

Very unintuitive strategy (letting all the differences spread), this is surely not the best path one could find for Grindahl. However, it is a very handy method in order to find a rather good candidate trail.

Truncated differential paths

Reducing the ”zoom” with truncated differentials allows to simplify the path search, but also decrease the probability that a good path exist in the search space. In general truncated differentials works well for byte-oriented primitives, not against bit-oriented, 32-bit or 64-bit hash functions.

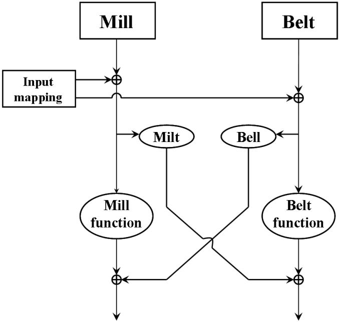
- **bit-oriented schemes** (e.g. RadioGatun): bit-wise diffusion will make the truncated differential analysis fail
- **byte-oriented schemes** (e.g. Grindahl): simplifies the path search while not reducing too much the search space
- **word-oriented schemes** (e.g. CubeHash): simplifies the path search too much, only very costly trails are likely to be found

Fugue presents security arguments regarding this kind of attacks.

Outline

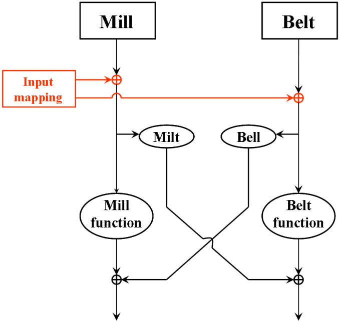
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
  - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
  - do **Milt**.
  - do **Bell**.
  - do **Mill function**.
  - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.



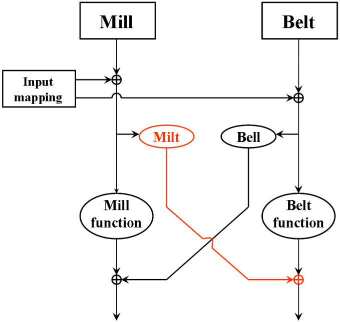
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
  - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
  - do **Milt**.
  - do **Bell**.
  - do **Mill function**.
  - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.



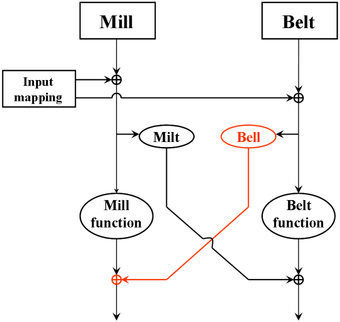
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
  - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
  - do **Milt**.
  - do **Bell**.
  - do **Mill function**.
  - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.



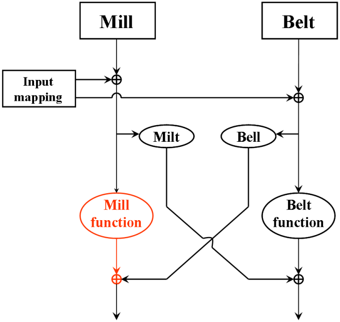
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
  - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
  - do **Milt**.
  - do **Bell**.
  - do **Mill function**.
  - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.



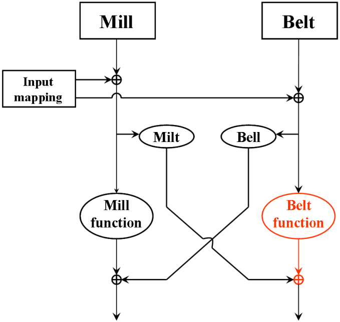
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
  - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
  - do **Milt**.
  - do **Bell**.
  - do **Mill function**.
  - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.

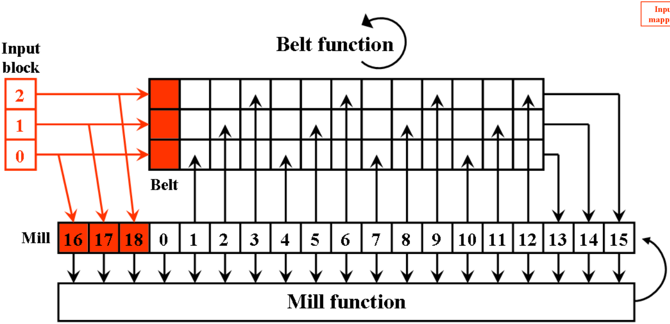


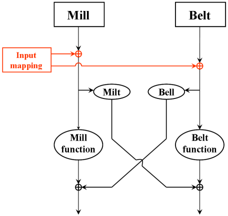
Symmetric differences: RadioGatun-32

- **initialize** the state with zeros.
- **for each round** do (while all the padded message hasn't been processed):
    - **XOR** 3 words of the Mill and 3 of the Belt to 3 new message words.
    - do **Milt**.
    - do **Bell**.
    - do **Mill function**.
    - do **Belt function**.
- do **16 blank rounds**.
- **do** (until we reach the good output size): a blank iteration and output 2 words from the Mill.

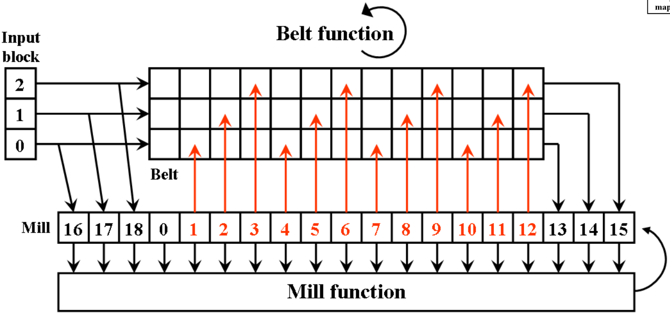


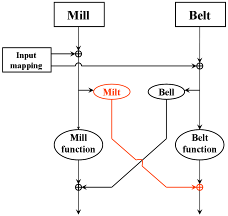
Symmetric differences: RadioGatun-32



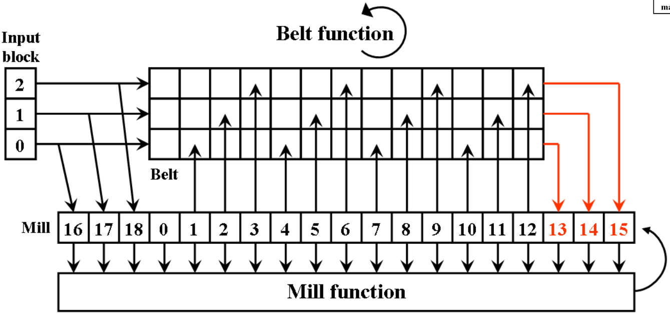


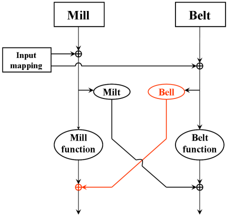
Symmetric differences: RadioGatun-32



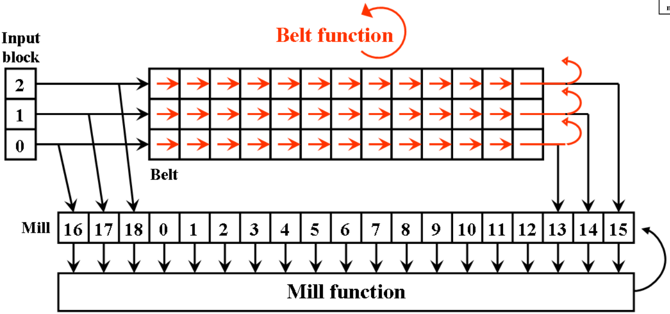


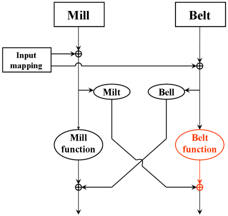
Symmetric differences: RadioGatun-32



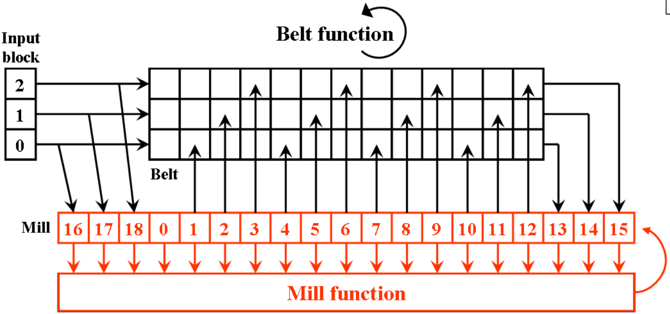


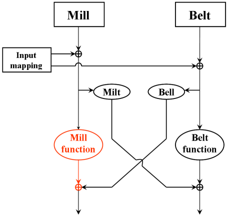
Symmetric differences: RadioGatun-32



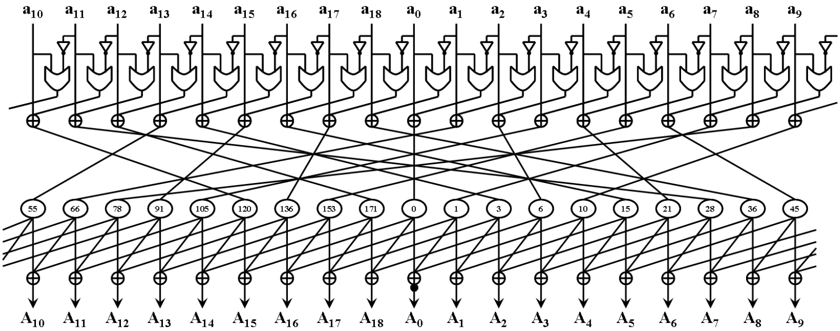


Symmetric differences: RadioGatun-32





Symmetric differences: RadioGatun-32



Symmetric differences

Consider **symmetric differences** for each word: only "all-different" or "equal".

- analysis REALLY simplified: you only have to study RadioGatun with w=1 (internal state of 58 bits).
- but each uncontrolled event cost a lot: all the complexity comes from the non-linear part in the Mill function.
- each event you want to force costs you one word of message freedom.
- the conditions can sometime be compressed (two same conditions on the same word).
- there may be contradicting conditions.

This techniques works well for bit-oriented primitives (RadioGatun [BDPV-RG06] [FP-FSE09] or PANAMA [RRPV-FSE01] [DV-FSE07])

Outline

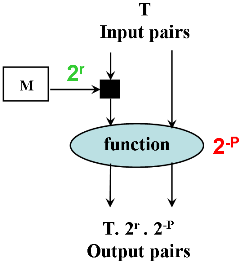
Using the freedom degrees

- **The techniques used are similar to the block-based hash functions: message modification, neutral bits, etc.** For example the control words used for attacking Grindahl ([P-AC07]) can be seen as byte-level message modifications.
- they lead to HUGE improvements. For example, from 2^{440} to 2^{112} for Grindahl [P-AC07].
- but because many freedom degrees can be used, and because the paths sometimes requires a few hundreds steps (RadioGatun [FP-FSE09]), one very often uses **automated tools** (Grindahl [P-AC07], RadioGatun [FP-FSE09], CubeHash [BKMP-AC09])
- Those tools are generally integrated directly during the path search (avoid the problem of ”good raw probability, but no freedom degrees possibility”)
- you can use structures (Grindahl or RadioGatun [K-SAC09]) or algebraic techniques (RadioGatun [BF-SAC08])

Using the freedom degrees: the trail backtracking cost

Trail backtracking [BDPV-RG06] is a method for estimating the cost of staying in a differential path when searching for collisions in hash functions.

- find a t-round trail starting and ending with no difference
- **Idea:**
  - start with T pairs at the beginning of the trail
  - each round i you go through, you have to ”pay” a probability P_i
  - each round i, you get r bits of freedom degrees



Using the freedom degrees: the trail backtracking cost

- The **number of valid pairs** at the end of a k-round trail is

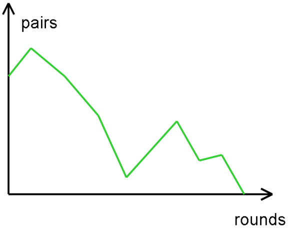
$$N(k) = T \times 2^{k \cdot r} \times 2^{-\sum_{i=1}^{k} P_i}$$

- at each round, you must have that the number of valid pairs is always ≥ 1 (can be removed if considering average cost), thus

$$T \geq \max_{0<k\leq t} \{2^{(\sum_{i=1}^{k} P_i) - k \cdot r}\}$$

- **total cost for the trail** is the sum of the number of pairs entering the rounds

$$cost = \sum_{j=1}^{t} N(j)$$



Using the freedom degrees: improving the trail backtracking

- Improvement: **find a good differential trail and define how you will use the freedom degrees at the same time.**
- **Search paths with a meet-in-the-middle tracking technique** (RadioGatun [FP-FSE09]):
  - keep track of the cost for forward paths (and cut off costly branches)
  - keep track of the cost for backward paths (and cut off costly branches)
  - meet-in-the-middle the two sets
  - adjust the costs during the meeting phase
- **Example:** for RadioGatun, with trail backtracking best attack found $2^{46 \cdot w}$ operations, with meet-in-the-middle tracking $2^{11 \cdot w}$ operations.
- to simplify the search, instead of randomly meeting in the middle, you can meet to a fixed difference (all-difference state for Grindahl [P-AC07]).

That’s all folks !

References

- **[ABMNP-ACISP09]:** J-P. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia and T. Peyrin, ”Inside the Hypercube”, ACISP 2009.
- **[BDPV-RG06]:** G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, ”The RadioGatún Hash Function Family”, second NIST Hash Workshop, 2006.
- **[BDPV-ECRYPTHW07]:** G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, ”Sponge Functions”, ECRYPT Hash Workshop, 2007.
- **[BDPV-EC08]:** G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, ”On the Indifferentiability of the Sponge Construction”, Eurocrypt 2008.
- **[BF-SAC08]:** C. Bouillaguet and P.A. Fouque, ”Analysis of the Radiogatun Hash Function”, SAC 2008.
- **[BKMP-AC09]:** E. Brier, S. Khazaei, W. Meier and T. Peyrin, ”Linearization Framework for Collision Attacks: Application to CubeHash and MD6”, Asiacrypt 2009.
- **[BP-ACNS09]:** E. Brier and T. Peyrin, ”Cryptanalysis of CubeHash”, ACNS 2009.
- **[CDMP-CRYPTO05]:** J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya, ”Merkle-Damgard Revisited: How to Construct a Hash Function”, CRYPTO 2005.
- **[D-SHA3list09]:** W. Dai, ”Collisions for CubeHash1/45 and CubeHash2/89”, NIST mailing list, 2008.

References

- **[DV-FSE07]:** J. Daemen and G. Van Assche, ”Producing Collisions for Panama, Instantaneously”, FSE 2007.
- **[FP-FSE09]:** T. Fuhr and T. Peyrin, ”Cryptanalysis of RadioGatun”, FSE 2009.
- **[GLP-AC08]:** M. Gorski, S. Lucks and T. Peyrin, ”Slide Attacks on a Class of Hash Functions”, Asiacrypt 2008.
- **[IP-FSE09]:** S. Indesteege and B. Preneel, ”Practical Collisions for EnRUPT”, FSE 2009.
- **[K-SAC09]:** D. Khovratovich, ”Cryptanalysis of hash functions with structures”, SAC 2009.
- **[MRH-TCC04]:** U. Maurer, R. Renner, and C. Holenstein, ”Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology”, TCC 2004.
- **[P-SHA3list09]:** T. Peyrin, ”Slide attacks on LUX”, NIST mailing list, 2008.
- **[P-AC07]:** T. Peyrin, ”Cryptanalysis of Grindahl”, Asiacrypt 2007.
- **[RRPV-FSE01]:** V. Rijmen, B. Van Rompay, B. Preneel and J. Vandewalle, ”Producing Collisions for PANAMA”, FSE 2001.

References: the hash functions

- **CubeHash:** D. Bernstein, ”CubeHash specifiation”, NIST SHA-3 competition candidate, 2008.
- **EnRupt:** S. O’Neil, K. Nohl, L. Henzen, ”EnRUPT Hash Function Specification”, NIST SHA-3 competition candidate, 2008.
- **FUGUE:** S. Halevi, W.E. Hall and C.S. Jutla, ”THE HASH FUNCTION FUGUE”, NIST SHA-3 competition candidate, 2008.
- **Grindahl:** L. Knudsen, C. Rechberger and S. Thomsen, ”Grindahl - a family of hash functions”, FSE 2007.
- **HAMSI:** Özgül Kücük, ”The hash function Hamsi”, NIST SHA-3 competition candidate, 2008.
- **Keccak:** G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, ”The Keccak sponge function family”, NIST SHA-3 competition candidate, 2008.
- **Luffa:** C. De Cannière, H. Sato and D. Watanabe, ”Hash function Luffa: Specification”, NIST SHA-3 competition candidate, 2008.
- **LUX:** I. Nikolic, A. Biryukov and D. Khovratovich, ”Hash family LUX”, NIST SHA-3 competition candidate, 2008.
- **PANAMA:** J. Daemen and C.S.K. Clapp, ”Fast hashing and stream encryption with Panama”, FSE 1998.
- **RadioGatun:** G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, ”The RadioGatún Hash Function Family”, second NIST Hash Workshop, 2006.
- **Shabal:** A.Canteaut, E. Bresson, B.Chevallier-Mames, C. Clavier, T. Fuhr, A.Gouget, T. Icart, J-F. Misarsky, M. Naya-Plasencia, P.Paillier, T.Pornin, J-R. Reinhard, C. Thuillet, M. Videau, ”Shabal, a Submission to NIST’s Cryptographic Hash Algorithm Competition”, NIST SHA-3 competition candidate, 2008.