A Family of Fast Syndrome Based Cryptographic Hash Functions

Daniel Augot, **Matthieu Finiasz** and Nicolas Sendrier

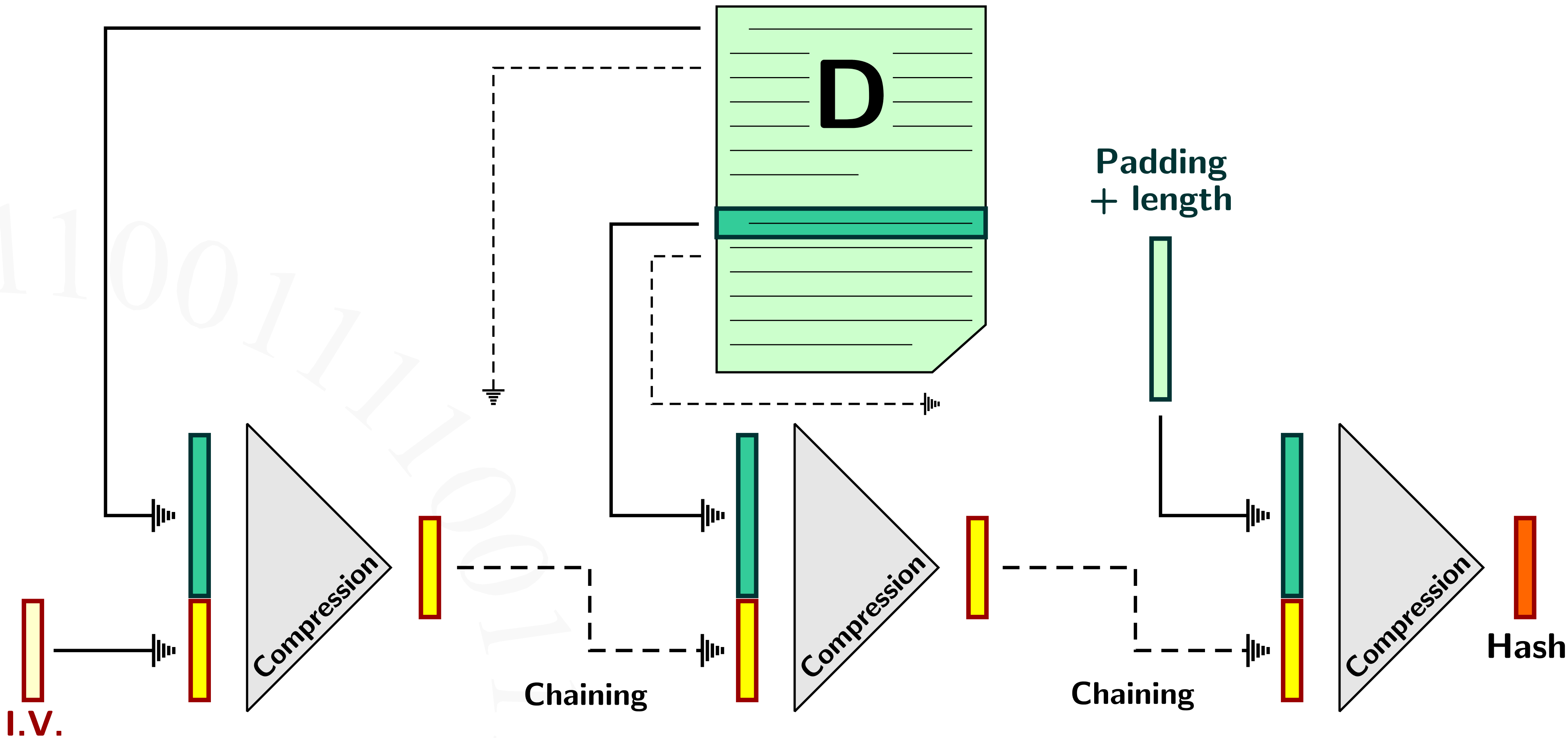
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

LASEC

Part I

General Facts about Hash Functions

The Merkle-Damgård construction



Recent discoveries

The chinese menace

- Many functions based on this construction are broken
  - MD4, MD5
  - RIPEMD
  - SHA-0, SHA-1
- Attacks inherent to this construction
  - Multicollisions [Joux - Crypto 04]
  - Second pre-image [Kelsey, Schneier - Eurocrypt 05]

⚠ Does not always behave like a random oracle.

Merkle-Damgård is not dead yet

- As long as collision resistance remains:
  - No multicollisions
  - No second preimage

- We wanted to build a hash function:
  - Provably collision resistant
  - Fast enough to compete with existing constructions

Part II

Description of the New Construction

The simplest compression function

Compress an input of s bits into r.

- Use a product by an $r \times s$ binary matrix

⚠ Linearity is bad: easy inversion!

The simplest compression function

Compress an input of s bits into r.

- Use a product by an $r \times s$ binary matrix

⚠ Linearity is bad: easy inversion!

- Code the input in a word of length n and given Hamming weight w, then multiply it by an $r \times n$ matrix

⚠ Constant weight encoding is slow!

The simplest compression function

Compress an input of s bits into r.

- Use a product by an $r \times s$ binary matrix

⚠ Linearity is bad: easy inversion!

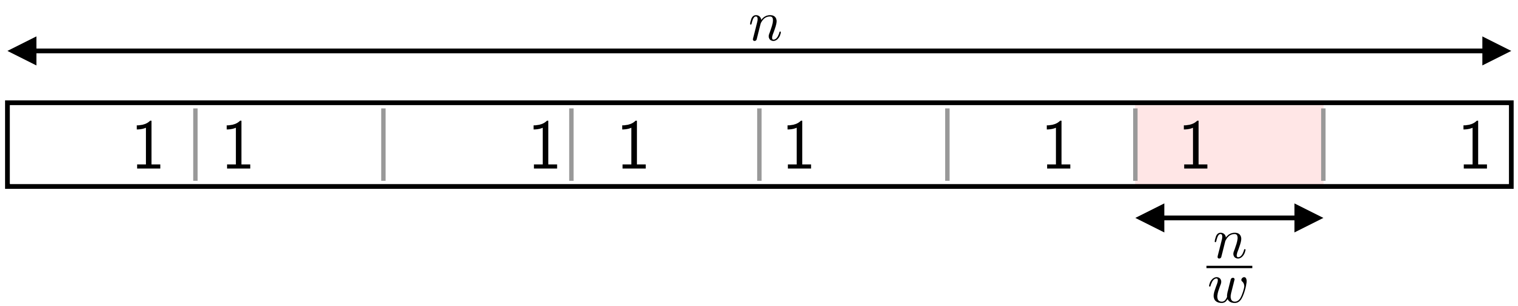
- Code the input in a word of length n and given Hamming weight w, then multiply it by an $r \times n$ matrix

⚠ Constant weight encoding is slow!

- Use a fast/lossy constant weight encoding technique.

Fast constant weight encoding

Using regular words



- We only consider regular words: words of weight w with one non-zero bit in each $\frac{n}{w}$ bits interval.
  - There are $\left(\frac{n}{w}\right)^w$ such words, thus $s = w \log_2 \left(\frac{n}{w}\right)$.

- With an exact encoding it would have been $s = \log_2 \binom{n}{w}$.

Step by step description

One round of the compression function

We use a random $r \times n$ binary matrix $\mathcal{H}$.

1. Concatenate the r chaining bits with $s - r$ bits from the document.
2. Split the s bits in w equal length strings s_i.
3. Convert each s_i in a column index h_i.
4. XOR the w columns h_i of $\mathcal{H}$.
5. Return the r-bit column obtained.

Part III

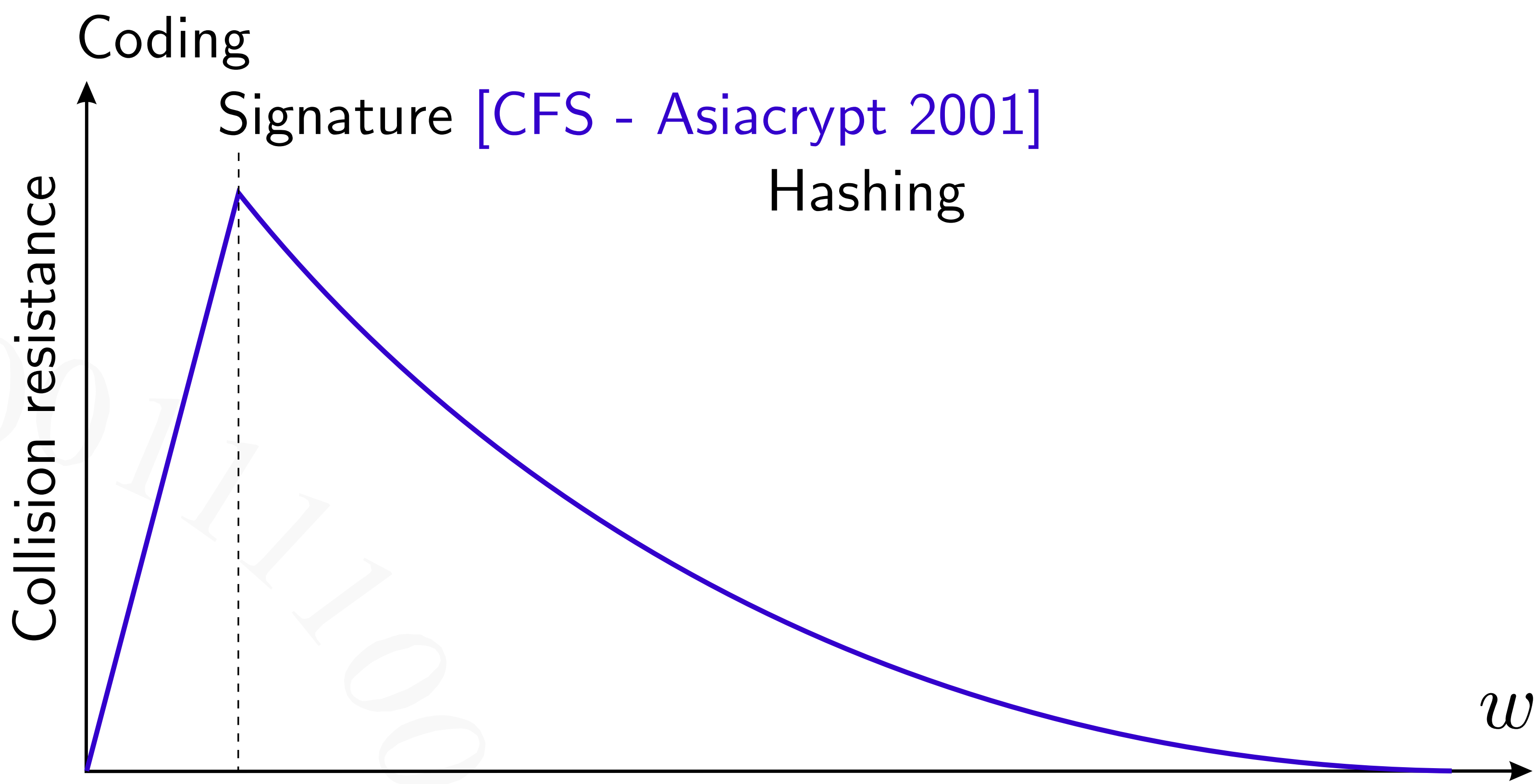
Security Analysis

- Inversion:
  - Given $\mathcal{S}$, find c of weight w such that $\mathcal{H} \times c = \mathcal{S}$.
- Collision:
  - Find c and c' of weight w such that $\mathcal{H} \times c = \mathcal{H} \times c'$.
  - Or find c of weight $< 2w$ such that $\mathcal{H} \times c = 0$.

- In both cases: solve an instance of Syndrome Decoding.
- With regular words, this problem is still NP-complete.

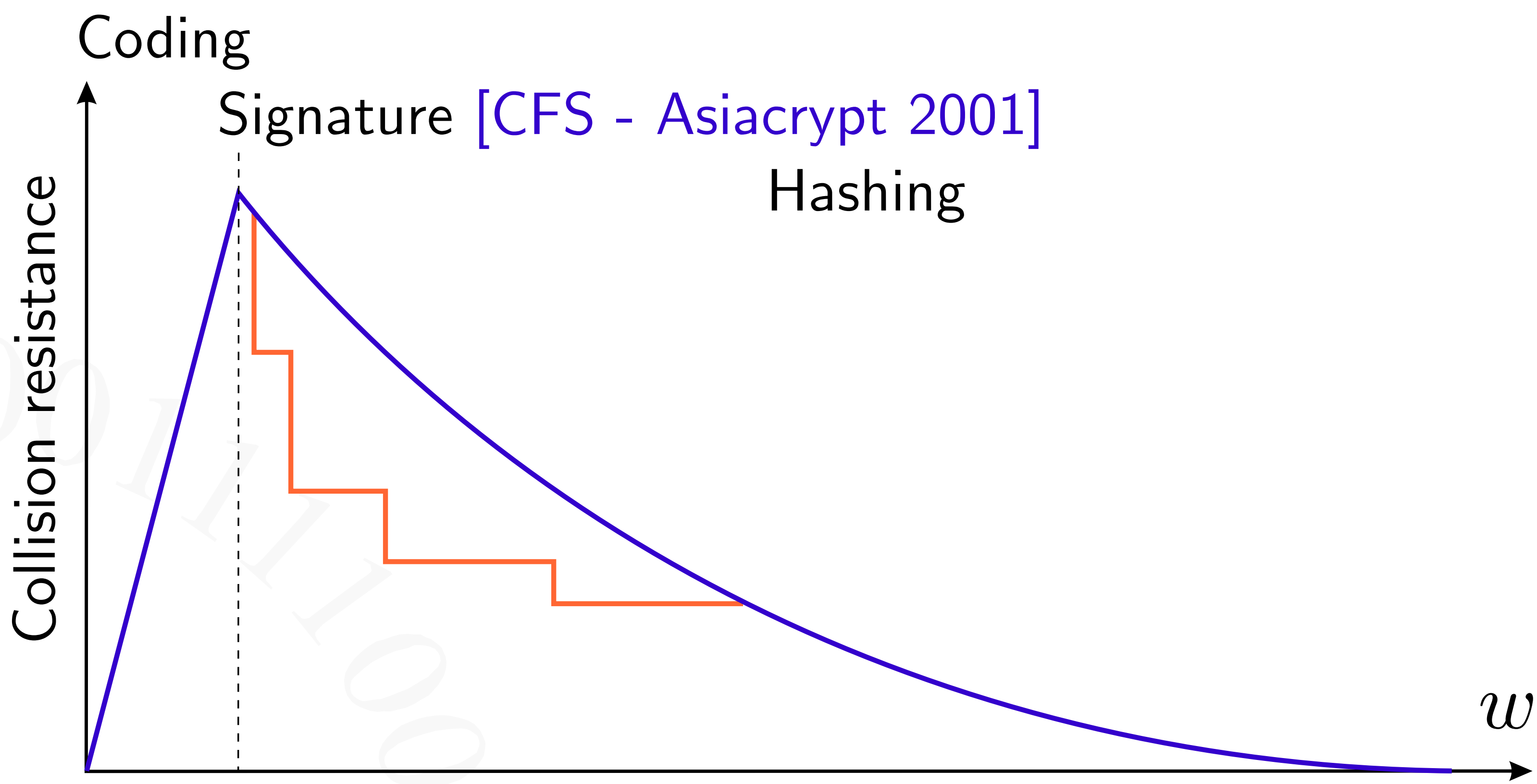
Practical security

Best known attacks



- Using classical decoding attacks [Canteaut, Chabaud 98].

Practical security
Best known attacks



- Using classical decoding attacks [Canteaut, Chabaud 98].
- Wagner's generalized birthday paradox [Coron, Joux 04].

The complexity of this attack depends of a parameter a.

- The attack can be applied for any a such that:

$$\frac{2^a}{a+1} \leq \frac{r}{w} \log_2 \left[\binom{\frac{n}{w}}{2} + 1 \right].$$

- Its complexity is $\mathcal{O}\left(2^{\frac{r}{a+1}}\right)$.

Attack complexity

Using the generalized birthday paradox

The complexity of this attack depends of a parameter a.

- The attack can be applied for any a such that:

$$\frac{2^a}{a+1} \leq \frac{r}{w} \log_2 \left[\binom{\frac{n}{w}}{2} + 1 \right].$$

- Its complexity is $\mathcal{O}\left(2^{\frac{r}{a+1}}\right)$.

It is crucial to keep a as small as possible!

- If we want compression it will always be possible to have $a = 4$.

Part IV

Choosing Suitable Parameters

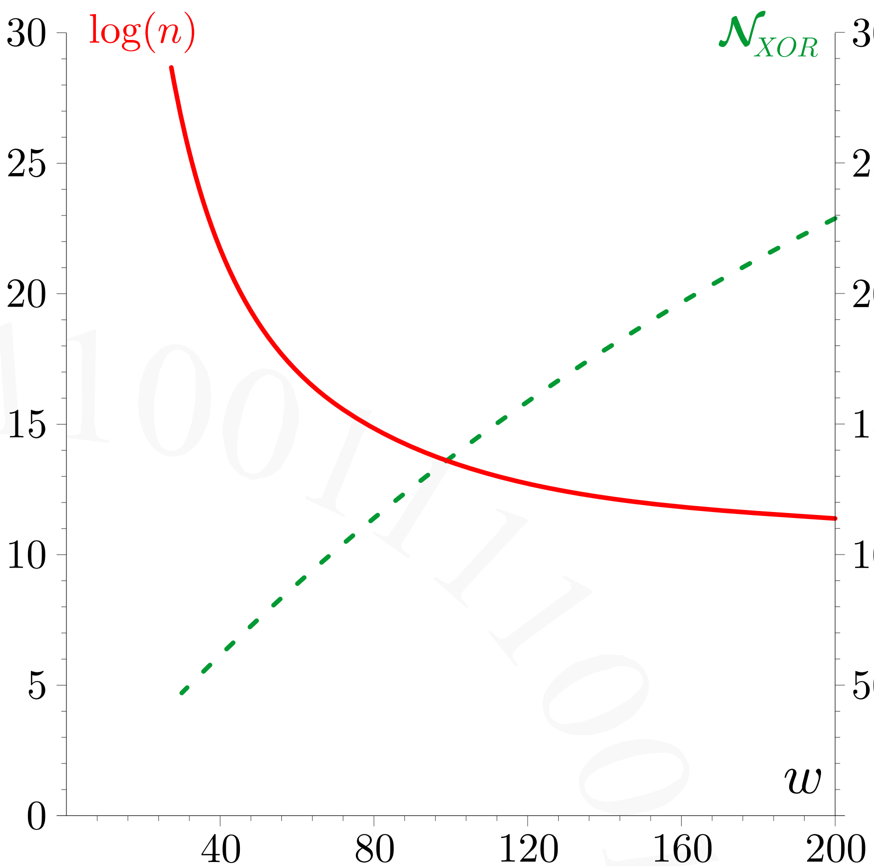
The only costly operations are binary XORs

- Speed will depend directly of the number $\mathcal{N}_{XOR}$ of binary XORs per input bit:

$$\mathcal{N}_{XOR} = \frac{rw}{w \log_2 \frac{n}{w} - r}.$$

- Faster for large values of n:
  - the larger $\mathcal{H}$, the faster the hashing.

Some suitable parameters



for: $r = 400$ *and* $a = 4$

$\log_2 \binom{n}{w}$	w	$\mathcal{N}_{XOR}$	size of $\mathcal{H}$
16	41	64.0	~ 1 Gbit
15	44	67.7	550 Mbits
14	47	72.9	293 Mbits
13	51	77.6	159 Mbits
12	55	84.6	86 Mbits
11	60	92.3	47 Mbits
10	67	99.3	26 Mbits
9	75	109.1	15 Mbits
8	85	121.4	8.3 Mbits
7	98	137.1	4.8 Mbits
6	116	156.8	2.8 Mbits
5	142	183.2	1.7 Mbits
4	185	217.6	1.1 Mbits

Obtained speed

- For $r = 400$, $w = 85$ and $\log_2 \frac{n}{w} = 8$
  - matrix size $\simeq 1$MB.
  - on a 2GHz P4 we get a throughput of 70Mbits/s.
- On a 64 bit CPU with 2MB cache
  - no more cache misses.
  - twice more binary XORs per CPU cycle.
  - throughput: *not tested*.

Part V

Possible Extensions

Reducing the output size

- If one wants an output shorter than 400 bits
  - Add a final transformation g.

- The function g takes r input bits and outputs r'
  - Used only once per hashing.
  - Can be more expensive than one standard round.
  - Possibly inefficient for short documents.

Online generation of $\mathcal{H}$

- Instead of using a truly random matrix $\mathcal{H}$, generate only required columns: $\mathcal{H}_i = f(i)$.
  - Possibility to use much larger matrices.
  - No more cache miss problems.

- What conditions should f verify for collision resistance?
  - Impossibility to find: $f(i_1) + \ldots + f(i_{2w}) = 0$.
  - If f is (as strong as) a block cipher we already have better constructions.

Conclusion

- We have "provable security".
  - No efficient generic attack.
- Throughput is high enough for most applications.
- Very wide parameter choice.
  - All parameters scale smoothly.
- Large outputs only.
  - Can be corrected via an output transformation.
- Uses more memory than other hash functions.
- Easy to implement!