The PHOTON Family of Lightweight Hash Functions

Jian Guo, Thomas Peyrin and Axel Poschmann

I2R and NTU

ECRYPT II Hash Workshop 2011

Tallinn, Estonia

Institute for Infocomm Research
A*STAR

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

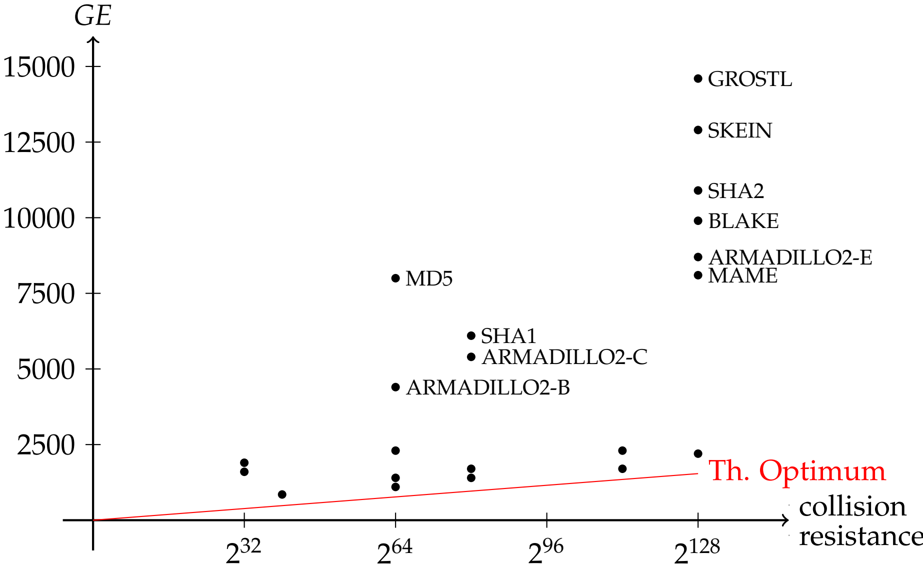
Outline

Lightweight hash functions

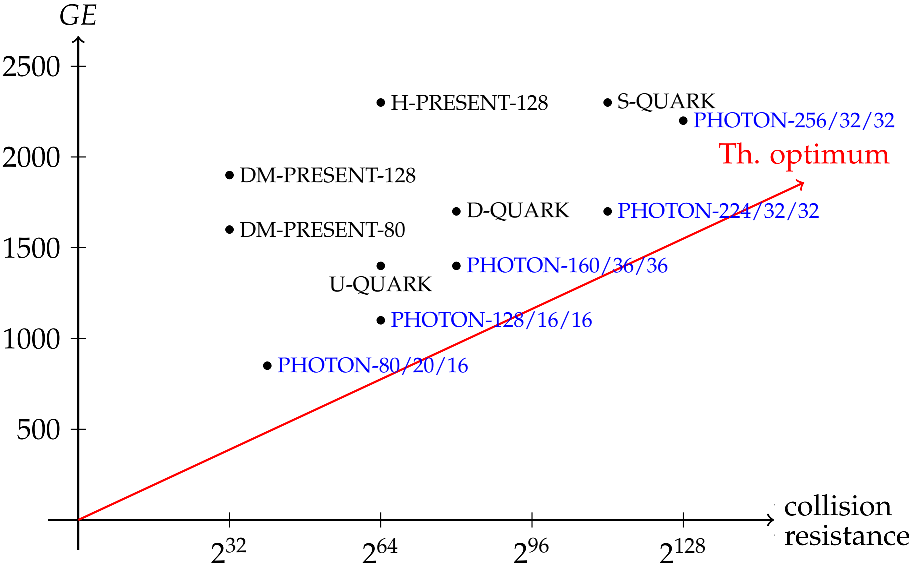
Why do we need lightweight hash functions ?

- RFID device authentication and privacy
- **in most of the privacy-preserving RFID protocols proposed, a hash function is required**
- a basic RFID tag may have a total gate count of anywhere from 1000-10000 gates, with **only 200-2000 gates** budgeted for security
- hardware throughput and software performances are not the most important criterias, but they must be acceptable

Current picture - graphically

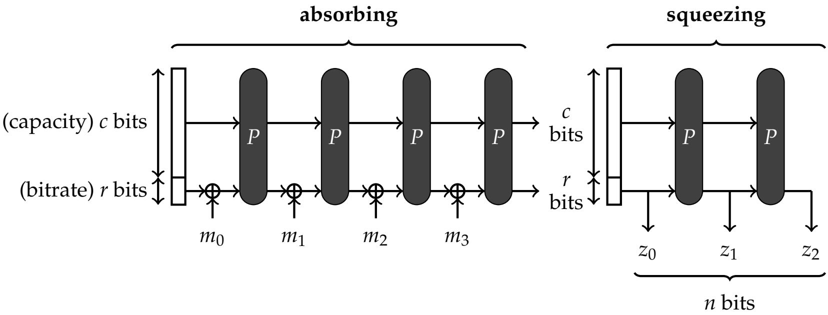


Current picture - graphically



Outline

Orginial sponge functions [Bertoni et al. 2007]



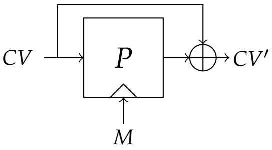
A sponge function has been proven to be indifferentiable from a random oracle up to $2^{c/2}$ calls to the internal permutation P. However, **the best known generic attacks have the following complexity:**

- **Collision:** $\min\{2^{n/2}, 2^{c/2}\}$
- **Second-preimage:** $\min\{2^{n}, 2^{c/2}\}$
- **Preimage:** $\min\{2^{\min\{n,c+r\}}, \max\{2^{\min\{n-r,c\}}, 2^{c/2}\}\}$

Sponges vs Davies-Meyer

We would like to build the smallest possible hash function with no better collision attack that generic ($2^{n/2}$ operations). Thus **we try to minimize the internal state size**:

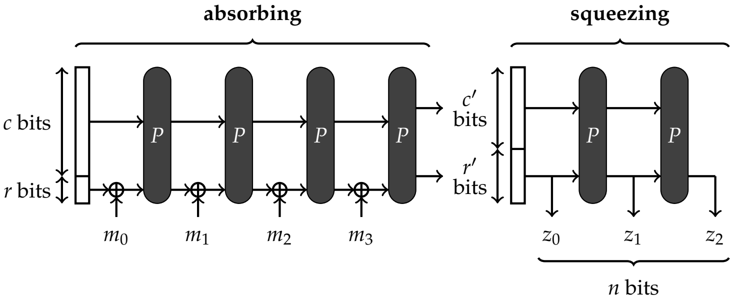
- **in a classical Davies-Meyer compression function** using a m-bit block cipher with k-bit key, one needs to store $2m + k$ bits. We minimize the internal state size with $m \simeq n$ and k as small as possible.



- **in sponge functions**, one needs to store $c + r$ bits. We minimize the internal state size by using $c \simeq n$ and a bitrate r as small as possible.

Sponge function will require about twice less memory bits for lightweight scenarios.

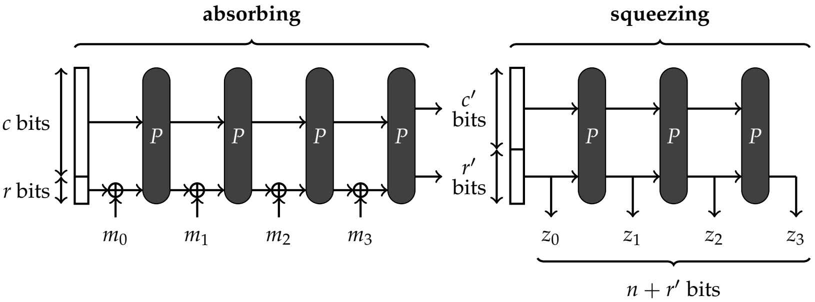
Generalization 1



Sponges with small r are slow for small messages (which is a typical usecase for lightweight applications, as an example EPC is 96 bit long). Thus **we can allow the output bitrate r' to be different from the input bitrate r** and obtain a preimage security / small message speed tradeoff:

- **Collision:** $\min\{2^{n/2}, 2^{c/2}\}$
- **Second-preimage:** $\min\{2^{n}, 2^{c/2}\}$
- **Preimage:** $\min\{2^{\min\{n,c+r\}}, \max\{2^{(\min\{n,c+r\}-r')}, 2^{c/2}\}\}$

Generalization 2



Sponges with $c \simeq n$ are not n-bit preimage resistant (often only preimage resistance is needed for lightweight applications). Thus **we can allow for bigger outputs by adding an extra squeezing step** and increase the preimage security:

- **Collision:** $\min\{2^{(n+r')/2}, 2^{c/2}\}$
- **Second-preimage:** $\min\{2^{(n+r')}, 2^{c/2}\}$
- **Preimage:** $\min\{2^{(\min\{n+r', c+r\})}, \max\{2^{\min\{n, c+r-r'\}}, 2^{c/2}\}\}$

Outline

MDS Matrix

What is an **MDS Matrix** (“Maximum Distance Separable”) ?

- it is used as **diffusion layer** in many block ciphers and in particular AES
- it has excellent diffusion properties. In short, **for a d-cell vector, we are ensured that at least $d+1$ input / output cells will be active** ...
- ... which is very good for linear / differential cryptanalysis resistance

The AES diffusion matrix can be implemented fast in software (using tables), but **the situation is not so great in hardware**. Indeed, even if the coefficients of the matrix minimize the hardware footprint, $d-1$ **cells of temporary memory are needed for the computation**.

$$A = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} =$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ \\ \vdots \\ \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \\ v'_0 \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Tweaking AES for hardware: AES-HW

The smallest AES implementation requires 2400 GE with 263 GE dedicated to the MixColumns layer (the matrix A is MDS).

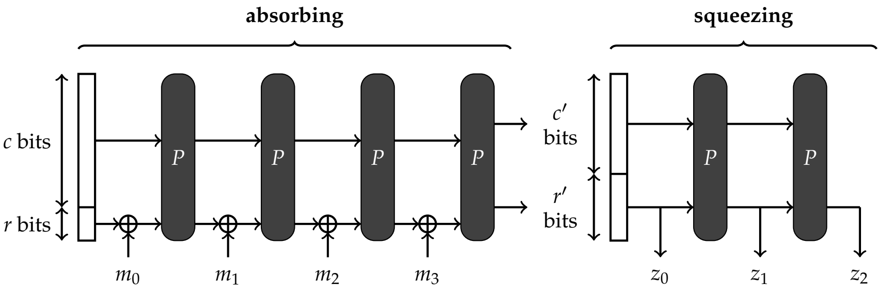
$$A = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}$$

Our tweaked AES-HW implementation requires 2210 GE with 74 GE dedicated to the MixColumnsSerial layer (the matrix $(B)^4$ is MDS):

$$(B)^4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix}^4 = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix} \qquad B^{-1} = \begin{pmatrix} 2 & 1 & 4 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Outline

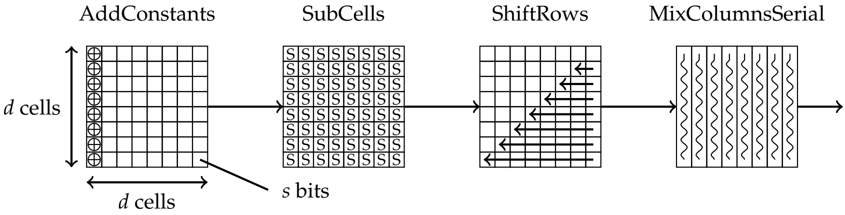
Domain extension algorithm



The $(c+r)$-bit internal state is viewed as a $d \times d$ matrix of s-bit cells.

PHOTON-$n/r/r'$		n	c	r	r'	d	s
PHOTON-80/20/16	P_{100}	80	80	20	16	5	4
PHOTON-128/16/16	P_{144}	128	128	16	16	6	4
PHOTON-160/36/36	P_{196}	160	160	36	36	7	4
PHOTON-224/32/32	P_{256}	224	224	32	32	8	4
PHOTON-256/32/32	P_{288}	256	256	32	32	6	8

Internal permutations



The internal permutations apply **12 rounds** of an AES-like fixed-key permutation:

- **AddConstants:** xor round-dependant constants to the first column
- **SubCells:** apply the PRESENT (when $s = 4$) or AES Sbox (when $s = 8$) to each cell
- **ShiftRows:** rotate the i-th line by i positions to the left
- **MixColumnsSerial:** apply the special MDS matrix to each columns

Outline

Extended sponge claims

Our security claims (a little bit more than flat sponge claims):

- **Collision:** $\min\{2^{n/2}, 2^{c/2}\}$
- **Second-preimage:** $\min\{2^{n}, 2^{c/2}\}$
- **Preimage:** $\min\{2^{\min\{n,c+r\}}, \max\{2^{\min\{n,c+r\}-r'}, 2^{c/2}\}\}$

For the security proofs, the internal permutation is modeled as a random permutation:

- the problem is reduced to studying the quality of the PHOTON internal permutations
- hermetic sponge strategy: it is assumed that the internal permutations have no structural flaw
- even if one finds a structural flaw for the internal permutations, it is unlikely to turn it into an attack ...
- ... **this is particularily true for** PHOTON **which has a very small bitrate** (i.e. the attacker has in practice a very small amount of freedom degrees in order to use the distinguisher).

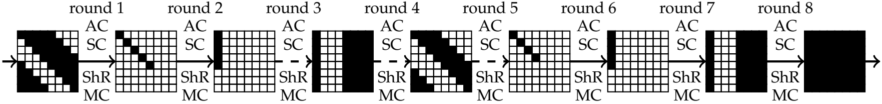
AES-like fixed-key permutation security

- AES-like permutations are simple to understand, well studied, provide very good security
- one can easily derive clear and powerful proofs on the minimal number of active Sboxes for 4 rounds of the permutation: $(d+1)^2$ **active Sboxes for 4 rounds of PHOTON**
- **we avoid any key schedule issue** since the permutations are fixed-key

	P_{100}	P_{144}	P_{196}	P_{256}	P_{288}
differential path probability	2^{-72}	2^{-98}	2^{-128}	2^{-162}	2^{-294}
differential probability	2^{-50}	2^{-72}	2^{-98}	2^{-128}	2^{-246}
linear approximation probability	2^{-72}	2^{-98}	2^{-128}	2^{-162}	2^{-294}
linear hull probability	2^{-50}	2^{-72}	2^{-98}	2^{-128}	2^{-246}

Table: Upper bounds for 4 rounds of the five PHOTON internal permutations.

Rebound attack and improvements



The currently best known technique achieves **8 rounds distinguishers** for an `AES`-like permutation, with quite low complexity.

	P_{100}	P_{144}	P_{196}	P_{256}	P_{288}
computations	2^8	2^8	2^8	2^8	2^{16}
memory	2^4	2^4	2^4	2^4	2^8
generic	2^{10}	2^{12}	2^{14}	2^{16}	2^{24}

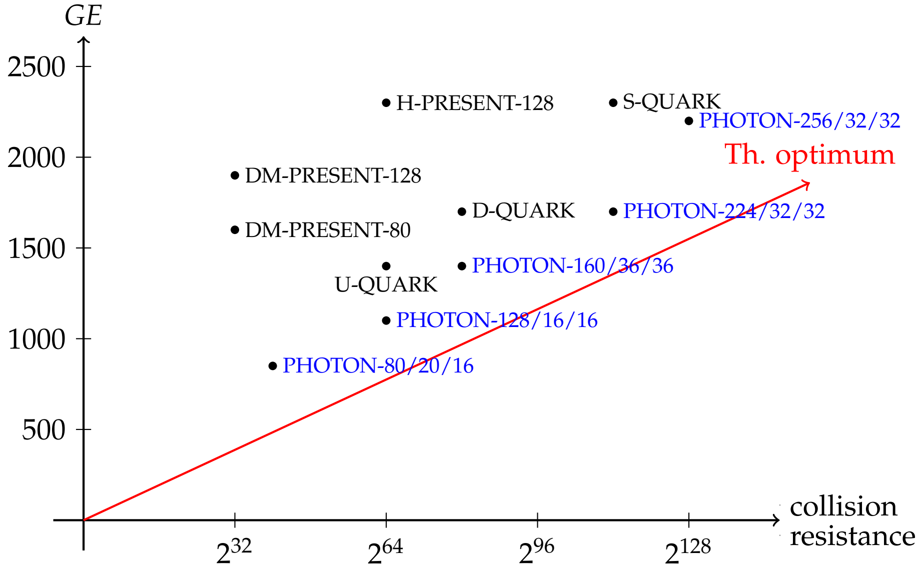
Improvements are unlikely since no key is used in the permutation, so **the amount of freedom degrees given to the attacker is limited to the minimum**.

Other cryptanalysis techniques

- **cube testers:** the best we could find within practical time complexity is at most 3 rounds for all PHOTON variants.
- **zero-sum partitions:** distinguishers for at most 8 rounds for the five proposed PHOTON variants (for complexity $\leq$ preimage claim).
- **algebraic attacks:** the entire system for the internal permutations of PHOTON consists of $d^2 \cdot N_r \cdot \{21, 40\}$ quadratic equations in $d^2 \cdot N_r \cdot \{8, 16\}$ variables.
- **slide attacks on permutation level:** all rounds of the internal permutation are made different thanks to the round-dependent constants addition.
- **slide attacks on operating mode level:** the sponge padding rule from PHOTON forces the last message block to be different from zero.
- **rotational cryptanalysis:** any rotation property in a cell will be directly removed by the application of the Sbox layer.
- **integral attacks:** can reach 7 rounds with complexity $2^{s(2d-1)}$.

Outline

Hardware implementation results



Conclusion

The PHOTON **family of hash functions**

- is very **simple**, clean, based on the AES design strategy
- are the **smallest hash functions** known so far
- provides acceptable software performances
- provides **provable security** against classical linear/differential cryptanalysis, and resists all known and recent attacks against hash functions with an extremly large security margin.

Latest results on https://sites.google.com/site/photonhashfunction/