ARMADILLO: a Multi-Purpose Cryptographic Primitive Dedicated to Hardware

Stéphane Badel[1], Nilay Dağtekin[1], Jorge Nakahara Jr[1], Khaled Ouafi[1], Nicolas Reffé[2], Pouyan Sepehrdad[1], Petr Sušil[1], Serge Vaudenay[1]

[1]EPFL, Lausanne, Switzerland

[2]Oridao, Montpellier, France

- small placental mammal, known for having a leathery **armor** shell.
- armadillo is Spanish for “little armored one”.
- Habitant: United States, from Texas to Illinois, Indiana and southern Ontario.
- used in the study of Leprosy: the few known non-human animal species that can contract the disease systemically.

ARMADILLO

- a general-purpose cryptographic function
  - I FIL-MAC: for challenge-response protocols
  - II Hashing and Digital Signatures
  - III PRNG and PRF
- hardware oriented
- target environments: RFID tags and sensor networks
- based on data-dependent permutations
- patent pending by Oridao (www.oridao.com)

Overall Design: **ARMADILLO2**

Input: initial value C, message block U_i
Output: $(V_c, V_t) = \textbf{ARMADILLO2}(C, U_i)$

- $X = Q(U_i, C\|U_i)$
- X undergoes a sequence of bit permutations, σ_0 and σ_1 and XOR with a constant, denoted by Q: maps a bitstring of k bits and a vector X of k bits into a vector of k bits, then
- $(V_c, V_t) = \textbf{ARMADILLO2}(C, U_i) = Q(X, C\|U_i) \oplus X$
- permutation Q defined recursively as

$$Q(s\|b, X) = Q(s, X_{\sigma_b} \oplus \gamma)$$

for $b \in \{0, 1\}$ and bitstrings s and X and a constant bitstring $\gamma = (10)^{k/2}$.

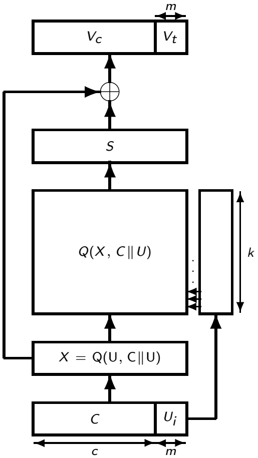


Figure: function of **ARMADILLO2**

Applications

I FIL-MAC: for challenge-response protocols

$$V_t = \mathbf{AMAC}_C(U)$$

II Hashing and Digital Signatures

$$V_c = \mathbf{AHASH}_{\mathrm{IV}}(\mathrm{message}\|\mathrm{padding})$$

III PRNG and PRF

$$\mathbf{APRF}_{\mathrm{seed}}(x) = \mathrm{head}_t(\mathbf{AHASH}_{\mathrm{seed}}(x\|\mathrm{cste}))$$

Old Design: **ARMADILLO**

Input: initial value C, message block U_i
Output: $(V_c, V_t) = \textbf{ARMADILLO}(C, U_i)$

- $\text{Xinter} = C \| U_i$
- $x = \overline{\text{Xinter}} \| \text{Xinter}$
- x undergoes a sequence of bit permutations, σ_0 and σ_1, denoted by P: maps a bitstring of k bits and a vector x of $2k$ bits into a vector of $2k$ bits, then

  $$S = P(Xinter, x) = \text{tail}_k((\overline{\text{Xinter}} \| \text{Xinter})_{\sigma_{\text{Xinter}}})$$

  where P is defined recursively as $P(s \| b, X) = P(s, X_{\sigma_b})$ for $b \in \{0, 1\}$ and bitstrings s and X and $P(\emptyset, X) = \text{tail}_k(X)$.
- $(V_c, V_t) \leftarrow S \oplus \text{Xinter}$

Schematic Diagram

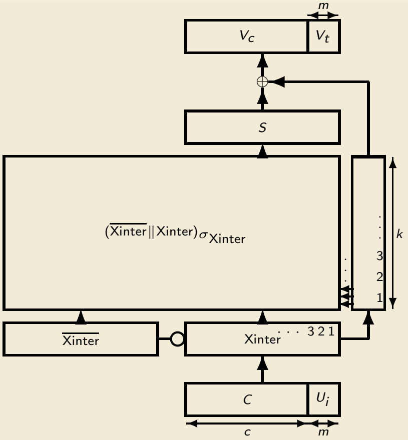


Figure: The **ARMADILLO** function

Table: Parameter vectors

Vector	k	c	m	r	t
A	128	80	48	6	128
B	192	128	64	9	192
C	240	160	80	10	240
D	288	192	96	12	288
E	384	256	128	15	384

ARMADILLO $\rightarrow$ ARMADILLO2

- **no** complementation of the k-bit input Xinter $= (C \| U_i)$.
- σ_i permutations (and so Q) operate on k-bit data $(C \| U_i)$.
- **no** truncation anymore at the output of Q.

ARMADILLO → ARMADILLO2

- **no** complementation of the k-bit input Xinter $= (C \| U_i)$.
- σ_i permutations (and so Q) operate on k-bit data $(C \| U_i)$.
- **no** truncation anymore at the output of Q.

more compact design and so performance advantage

1. ARMADILLO
2. Parameters
3. Security
4. Hardware
5. Conclusions

Defense Mechanism

Greek Ouroboros

Figure: Armadillo Lizard

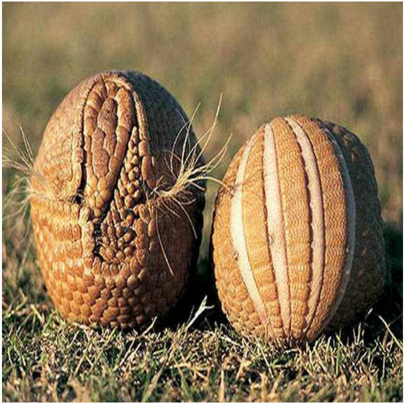
Security Bounds

- characterized by two parameters S_{offline} and S_{online}.
- the best offline attack has complexity $2^{S_{\text{offline}}}$.
- the best online attack, with practical complexity, has success probability $2^{-S_{\text{online}}}$.
- aim at $S_{\text{offline}} \geq 80$ and $S_{\text{online}} \geq 40$, but we can only upper bound S_{offline} and S_{online}.

Security Concern

- attack against **ARMADILLO**, but **not ARMADILLO2**.
- extra pre-processing in **ARMADILLO2**, i.e $X = Q(U_i, C \| U_i)$ prevents the attack on **ARMADILLO**.

Armored Armadillo



Hardware implementation of the **ARMADILLO** function

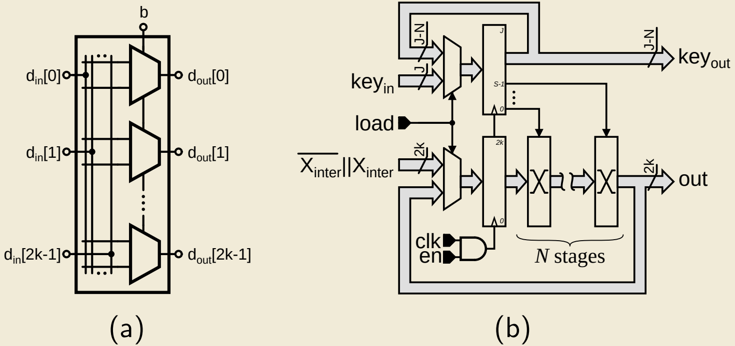


Figure: (a) one permutation stage, (b) *P* function building block

Is FOM = throughput / GE enough?

Is FOM = throughput / GE enough?

- at first sight: provides a more general measure of quality.

Is FOM = throughput / GE enough?

- at first sight: provides a more general measure of quality.
- **BUT**, it is flawed: how about trading off throughput for power?

Is FOM = throughput / GE enough?

- at first sight: provides a more general measure of quality.
- **BUT**, it is flawed: how about trading off throughput for power?
- according to this metric, two designs A and B are as efficient:
  - A's throughput and area is twice B's throughput and area.
  - B's power dissipation is half that of A.

Is FOM = throughput / GE enough?

- at first sight: provides a more general measure of quality.
- **BUT**, it is flawed: how about trading off throughput for power?
- according to this metric, two designs A and B are as efficient:
  - A's throughput and area is twice B's throughput and area.
  - B's power dissipation is half that of A.
  - by doubling B's operating frequency, its throughput can be made equal to that of A while consuming the same power and still occupying a smaller area.
  - B should be recognized as superior to A.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.
  - depends heavily on the process technology, the supply voltage, and the parasitic capacitances.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.
  - depends heavily on the process technology, the supply voltage, and the parasitic capacitances.
  - vary largely depending on the method used to measure it.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.
  - depends heavily on the process technology, the supply voltage, and the parasitic capacitances.
  - vary largely depending on the method used to measure it.
  - different standard-cell libraries exhibit various power/area/speed trade-offs.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.
  - depends heavily on the process technology, the supply voltage, and the parasitic capacitances.
  - vary largely depending on the method used to measure it.
  - different standard-cell libraries exhibit various power/area/speed trade-offs.
  - generally not available.

How about FOM = throughput / GE^2?

- Solution: dividing the metric by the power dissipation.
- **But** it has its own problems:
  - power dissipation: an extremely volatile quantity.
  - subject to the same error factors as the area.
  - depends heavily on the process technology, the supply voltage, and the parasitic capacitances.
  - vary largely depending on the method used to measure it.
  - different standard-cell libraries exhibit various power/area/speed trade-offs.
  - generally not available.
- A fairer FOM: include the influence of power dissipation.
- we can assume that the power is proportional to the gate count.

- T-stage pipeline: $R = k/(N.T)$ of permutations at each stage.
- throughput: $1/R$ items per cycle and the latency: k/N cycles.
- more pipeline stages: more hardware replication, area, power.
  - a fully serial implementation: ideal for RFID applications.
- In practice, to maximize FOM with $T = 1$, we obtain $N = 4$ for **ARMADILLO2**.
- obtain an area of 4 030 GE, 77 μW, and a latency of 44 cycles (1.09 Mbps for hashing or 2.9 Mbps for encryption).

Synthesis Results

- synthesis in a 0.18μm CMOS process using a commercial standard-cell library.
- Synopsys Design Compiler in topographical mode.
- power consumption: Synopsys Primetime-PX using gate-level vector-based analysis.

Synthesis results at 1 MHz

Algorithm	Vect.	N=1				N=4			
		Area (GE)	Power (μW)	Thr. (kbps)	Latency (cycles)	Area (GE)	Power (μW)	Thr. (kbps)	Latency (cycles)
ARMADILLO2	**A**	2 923	44	272	176	4 030	77	1 090	44
	B	4 353	65	250	256	6 025	118	1 000	64
	C	5 406	83	250	320	7 492	158	1 000	80
	D	6 554	102	250	384	8 999	183	1 000	96
	E	8 653	137	250	512	11 914	251	1 000	128

Table: the throughput values correspond to hash mode.

Comparing hash functions performance at 100 kHz

Algorithm	Digest (bits)	Block (bits)	Area (GE)	Time (cycles/block)	Throughput (kb/s)	Logic (μm)	FOM (nanobit/cycle.GE^2)
ARMADILLO2-A	80	48	4 030	44	109	0.18	**67.17**
ARMADILLO2-A	80	48	2 923	176	27	0.18	**31.92**
H-PRESENT-128	128	128	4 256	32	200	0.18	110.41
ARMADILLO2-B	128	64	6 025	64	1000	0.18	**27.55**
MD4	128	512	7 350	456	112.28	0.13	20.78
ARMADILLO2-B	128	64	4 353	256	250	0.18	**13.19**
MD5	128	512	8 400	612	83.66	0.13	11.86
ARMADILLO2-C	160	80	7 492	80	100	0.18	**17.81**
ARMADILLO2-C	160	80	5 406	320	250	0.18	**8.55**
SHA-1	160	512	8 120	1 274	40.18	0.35	6.10
ARMADILLO2-D	192	96	8 999	96	100	0.18	**12.35**
C-PRESENT-192	192	192	8 048	108	59.26	0.18	9.15
ARMADILLO2-D	192	96	6 554	384	25	0.18	**5.82**
MAME	256	256	8 100	96	266.67	0.18	40.64
ARMADILLO2-E	256	128	11 914	128	100	0.18	**7.05**
SHA-256	256	512	10 868	1 128	45.39	0.35	3.84
ARMADILLO2-E	256	128	8 653	512	25	0.18	**3.34**

Comparing performance of ciphers at 100 kHz

Algorithm	Key (bits)	Block (bits)	Area (GE)	Time (cycles/block)	Throughput (kb/s)	Logic (μm)	FOM (nanobit/cycle.GE^2)
DES	56	64	2 309	144	44	0.18	83.36
PRESENT-80	80	64	1 570	32	200	0.18	811.39
Grain	80	1	1 294	1	100	0.13	597.22
KTANTAN64	80	64	927	128	50	0.13	581.85
KATAN64	80	64	1 269	85	75	0.13	467.56
ARMADILLO2-A	80	128	4 030	44	291	0.18	**179.12**
Trivium	80	1	2 599	1	100	0.13	148.04
PRESENT-80	80	64	1 075	563	11	0.18	98.37
ARMADILLO2-A	80	128	2 923	176	73	0.18	**85.12**
mCrypton	96	64	2 681	13	500	0.13	684.96
PRESENT-128	128	64	1 886	32	200	0.18	562.27
HIGHT	128	64	3 048	34	189	0.25	202.61
TEA	128	64	2 355	64	100	0.18	180.31
ARMADILLO2-B	128	192	6 025	64	300	0.18	**82.64**
ARMADILLO2-B	128	192	4 353	256	75	0.18	**39.58**
AES-128	128	128	3 400	1 032	12	0.35	10.73
ARMADILLO2-C	160	240	7 492	80	300	0.18	**53.45**
ARMADILLO2-C	160	240	5 406	320	75	0.18	**25.66**
DESXL	184	64	2 168	144	44	0.18	94.56
ARMADILLO2-D	192	288	8 999	96	300	0.18	**37.04**
ARMADILLO2-D	192	288	6 554	384	75	0.18	**17.46**
ARMADILLO2-E	256	384	11 914	128	300	0.18	**21.13**
ARMADILLO2-E	256	384	8 653	512	75	0.18	**10.02**

1. ARMADILLO
2. Parameters
3. Security
4. Hardware
5. Conclusions

- a new hardware dedicated cryptographic function design.
- two instances: **ARMADILLO** and **ARMADILLO2**
- **ARMADILLO2**: fully serial architecture, 2 923 GE could perform one compression within 176 clock cycles, consuming 44 μW at 1 MHz.
- another tradeoff leads us to 4 030 GE, 44 cycles, 77 μW, 11 Mbps of hashing, and 2.9 Mbps of encryption.

Questions?