Hardware Implementation and Side-Channel Analysis of LAPIN

L. Gaspar, G. Leurent, F.X. Standaert

UCL Crypto Group, Universit catholique de Louvain

Fréjus, France, June 24th, 2013

Outline

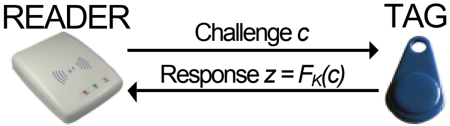
1. Introduction
2. Lapin protocol
3. Implementation
4. Performance evalution
5. Side-channel analysis
6. Conclusion

Outline

1. Introduction
2. Lapin protocol
3. Implementation
4. Performance evalution
5. Side-channel analysis
6. Conclusion

Light-weight Shared-key Authentication Protocols

- Lightweight shared-key authentication protocols are widely used
- Typical settings:
  1. Reader generates challenge c
  2. Tag computes response $z = F_K(c)$
  3. Reader computes $z' = F_K(c)$
  4. Reader accepts the Tag if $z = z'$



Ideal Authentication Protocol

Considered conditions:

- Protocol properties:
  1. Provably secure
  2. Small amount of transfered data
  3. Minimum of rounds (i.e. 2)
  4. Fast response (low latency)
- Tag properties:
  1. Small footprint (in HW)
  2. Small code size (in SW)
  3. Low-power
  4. Low-cost

Protocol Classification

Many such algorithms exist, e.g.:

- Block-cipher based schemes
  - AES-based – may be too heavy for some appl.
  - Present-based – more suitable
- Schemes based on hardness of a mathematical problem:
  - Learning Parity with Noise problem (LPN)
    - Hopper-Blum protocol (HB)
    - Variants of HB (HB+, HB-MP, etc.)
    - Lapin protocol [1]
  - Others

[1] *Lapin: an efficient authentication protocol based on Ring-LPN*, S. Heyse, E. Kiltz, V. Lyubashevsky, Ch. Paar, K. Pietrzak, pages 346-365, FSE 2012

Learning Parity with Noise Problem (LPN)

- Given a set of samples $(A, t = A \cdot s + e)$ with a random error e, where t,$e \in \mathbb{F}_2^n$ and $A \in \mathbb{F}_2^{n \times n}$
- Find the secret $s \in \mathbb{F}_2^n$
- Solution:
  a) if $\mathbf{e} = \mathbf{0}$ than Gaussian elimination can solve it
  $\rightarrow$ **no security!**
  b) if $\mathbf{e} > \mathbf{0}$ than it may become an NP-Hard problem
  $\rightarrow$ **suitable for cryptography!**

Note: The error e is generated with the Bernoulli distribution with parameter τ. HW(e) $\approx n\tau$

Ring-LPN problem

- Ring Learning Parity with Noise (Ring-LPN) is an extension of LPN to rings
- The matrix A has a special structure. This way $A \cdot s$ is equivalent to the multiplication in the ring $\mathrm{R} = \mathbb{F}_2[X]/f(X)$
- **Lapin provably secure based on the Ring-LPN problem**

Outline

Lapin Protocol Parameters

- 2-round protocol
- Public parameters:

R, n	ring $\mathsf{R} = \mathbb{F}_2[X]/f(X)$, $\deg(f) = n$
λ	security level parameter (in bits)
π	mapping $\{0,1\}^\lambda \rightarrow \mathsf{R}$
$\tau \in (0, 1/2)$	parameter of Bernoulli distribution
$\tau' \in (\tau, 1/2)$	reader acceptance threshold

- Secret parameter:

$K = (s, s')$	shared secret key, while $(s, s') \xleftarrow{\$} \mathsf{R}$

Lapin Protocol description

Public parameters: R, π:$\{0,1\}^\lambda \to$ R, τ, τ', λ
Secret key: $K = (s, s') \in$ R^2

	Tag		**Reader**
①		$\xleftarrow{c}$	$c \xleftarrow{\$} \{0,1\}^\lambda$
②	$r \xleftarrow{\$}$ R*; $e \xleftarrow{\$}$ Ber$_\tau^{\mathrm{R}} \in$ R		
③	$z := r \cdot (s \cdot \pi(c) + s') + e$	$\xrightarrow{(r,z)}$	
④			if $r \notin$ R* reject
⑤			$e' := z - r \cdot (s \cdot \pi(c) + s')$
⑥			if $HW(e') > n \cdot \tau'$ reject else accept

Masking countermeasure

- **Objective:** decrease the correlation between the consumed power and the processed sensitive data
- **Implementation:** all sensitive variables must be split to shares and computations should be performed on each share separately (if possible)
- **Conditions** for effective masking:
  - the leakage of each share is independent from the others
  - sufficient noise is present in the device

Example:

$$\begin{aligned} h_1 &= q_1 \\ &\vdots \\ h_d &= q_d \\ h_{d+1} &= h \oplus \bigoplus_{i=1}^{d} q_i \end{aligned}$$

Masking of Lapin

1. Split sensitive variables s, s' and e into $d+1$ shares:

$$\begin{aligned} s &= s_1 \oplus s_2 \oplus \cdots \oplus s_{d+1}, \\ s' &= s'_1 \oplus s'_2 \oplus \cdots \oplus s'_{d+1}, \\ e &= e_1 \oplus e_2 \oplus \cdots \oplus e_{d+1} \end{aligned}$$

2. Derive a formula allowing to demask the output

$$\begin{aligned} z &= (\pi(c) \cdot s \oplus s') \cdot r \oplus e \\ &= \left[\pi(c) \cdot (s_1 \oplus \cdots \oplus s_{d+1}) \oplus (s'_1 \oplus \cdots \oplus s'_{d+1})\right] \cdot r \oplus (e_1 \oplus \cdots \oplus e_{d+1}) \\ &= [(\pi(c) \cdot s_1 \oplus s'_1) \cdot r \oplus e_1] \oplus \cdots \oplus \left[(\pi(c) \cdot s_{d+1} \oplus s'_{d+1}) \cdot r \oplus e_{d+1}\right] \\ &= z_1 \oplus \cdots \oplus z_{d+1} \end{aligned}$$

- **Lapin is linear** = each share is computed **separately**

Outline

Definition of constants

Constants are chosen as in the Lapin paper:

- $deg(f(X)) = n = 621$
- $m = 5$
- m factors of $f(X)$ are:
- $\tau = 1/6$
- $\tau' = 0.29$
- $\lambda = 80$ bits

$$
\begin{array}{lllllllllll}
f_1(X) & = & X^{127} & + & X^8 & + & X^7 & + & X^3 & + & 1 \\
f_2(X) & = & X^{126} & + & X^9 & + & X^6 & + & X^5 & + & 1 \\
f_3(X) & = & X^{125} & + & X^9 & + & X^7 & + & X^4 & + & 1 \\
f_4(X) & = & X^{122} & + & X^7 & + & X^4 & + & X^3 & + & 1 \\
f_5(X) & = & X^{121} & + & X^8 & + & X^5 & + & X & + & 1
\end{array}
$$

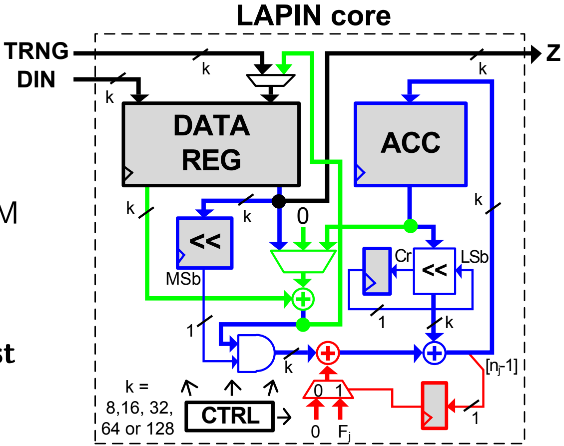
$=>$ 128-bit datapath is suitable, since $deg(f_j(x)) < 128$

Polynomial multiplication & reduction

- We have implemented a **128-bit "school-book" polynomial multiplication unit** because:
  - it can be performed **in parallel with 1-bit reduction**
  - its hardware implementation is **very small**
  - its implementation can operate on **high frequencies**
- This **unit can be shared** for Lapin computations as well as error e transformation

Implementation description

- 8b to 128b datapath width
- Data registers in RAM
- Accumulator in RAM
- Carry register if $k < 128$
- **Shift register must not load sensitive data**



Multiplication, reduction and addition

Outline

Cost evaluation & Timing results

- Lapin was synthesized for Xilinx Virtex 5 FPGA

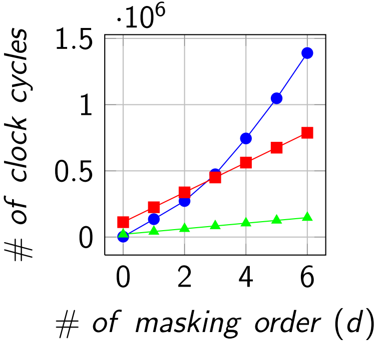
	One share ($d = 0$)					Three shares ($d = 2$)				
	8-bit	**16-bit**	**32-bit**	**64-bit**	**128-bit**	**8-bit**	**16-bit**	**32-bit**	**64-bit**	**128-bit**
F_1[cyc]	4,048	2,024	1,012	506	257	12,144	6,072	3,036	1,518	771
F_2[cyc]	4,160	2,080	1,040	520	264	12,480	6,240	3,120	1,560	792
F_3[cyc]	4,208	2,104	1,052	526	267	12,624	6,312	3,156	1,578	801
F_4[cyc]	4,224	2,112	1,056	528	268	12,672	6,336	3,168	1,584	804
F_5[cyc]	4,336	2,168	1,084	542	275	13,008	6,504	3,252	1,626	825
TOTAL[cyc]	**20,977**	**10,489**	**5,245**	**2,623**	**1,332**	**62,961**	**31,481**	**15,741**	**7,871**	**3,996**
Slices	170	214	254	294	414	213	232	311	330	451
BRAM 18kb	2	2	1	0	0	2	2	1	0	0
BRAM 36kb	0	0	1	3	6	0	0	1	3	6
f_{MAX}[MHz]	139.7	141.9	145.4	147.2	163.5	125.3	127.5	127.2	130.2	140.3

- $d = 0$: Lapin without masking
- $d = 2$: Masked Lapin – secure to second-order attacks

Comparison

# of m. order d	AES (SW) [Our]	Lapin[a] (SW) [Heyse2012]	Lapin (HW) [Our]
0	3000	112500	20977
1	135087	225016	41969
2	272159	337532	62961
3	474287	450048	83953
4	744769	562564	104945
5	1047878	675080	125937
6	1389780	787596	146929

[a] For $d > 0$ values are estimated



- When increasing d, number of clock cycles grows **linearly for Lapin** and **quadratically for AES**

=> **It's substantially cheaper to increase security of Lapin to higher-order SCA than of AES**

Outline

1. Introduction
2. Lapin protocol
3. Implementation
4. Performance evalution
5. Side-channel analysis
6. Conclusion

Attack model

- **Target operation:** $s \cdot \pi(c)$, where π is zero padding
- **Assumption:** Device leaks accumulator H. weight
- Accumulator is updated during the multiplication loop:

$$a_0 = 0 \qquad\qquad a_{i+1} \leftarrow \begin{cases} 2 \cdot a_i + s & \text{if } c[80-i] = 1 \\ 2 \cdot a_i & \text{otherwise} \end{cases}$$

- The value of a after a few cycles of computation is a small multiple of the secret:

$$a_{80} = s \cdot c \qquad\qquad a_i = s \cdot \overbrace{\sum_{j=1}^{i} c[80-j] X^{i-j}}^{m_i(c)}$$

- Device leaks $\mathbf{HW(a_i)}$

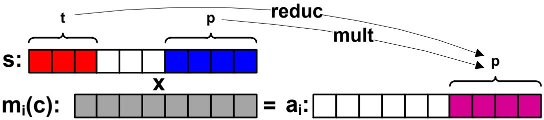
Unprotected design ($d = 0$)

Two options:

- Attack can target **several clock cycles in a single trace** with the **same challenge** c
- Attack can target the **same clock cycle in several traces**, while **challenges are chosen** appropriatelly

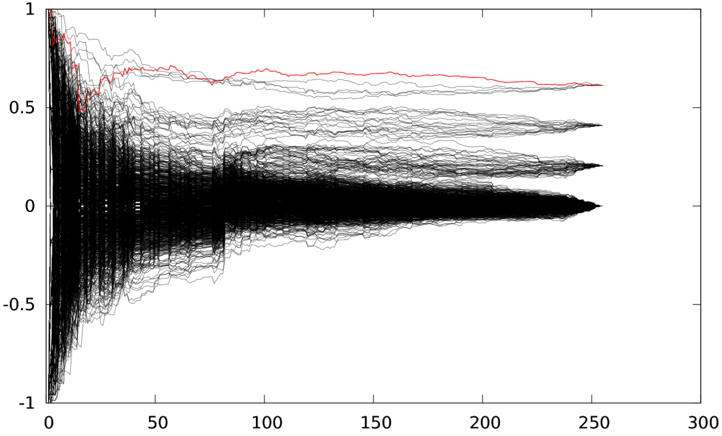
Attack:

- Predict some bits of $a_i = s \cdot m_i(c)$
- If $deg(a_i) \leq t$ we can compute **p least significant bits** of a_i from the p *least* significant and t *most* significant bits of s.



Unprotected design ($d = 0$)

- Correlation for $t = 7$ and $p = 3$



Unprotected design ($d = 0$)

- **Other approach:** Prediction of modular reduction impact on hamming weight
- Assumption: accumulator contains value α that will be shifted and reduced in next clock cycle

$$\alpha \cdot X \bmod f = \begin{cases} (\alpha \ll 1) & \text{if } \mathrm{MSb}(\alpha) = 0 \\ (\alpha \ll 1) \oplus \bar{f} & \text{if } \mathrm{MSb}(\alpha) = 1 \end{cases}$$

Since the polynomials f are pentanomials, we have $\mathrm{HW}(\bar{f}) = 4$, and

$$\mathrm{HW}(\alpha \cdot X \bmod f) = \begin{cases} \mathrm{HW}(\alpha) & \text{if } \mathrm{MSb}(\alpha) = 0 \\ \mathrm{HW}(\alpha) + \{\pm 1, \pm 3\} & \text{if } \mathrm{MSb}(\alpha) = 1 \end{cases}$$

Outline

1. Introduction
2. Lapin protocol
3. Implementation
4. Performance evalution
5. Side-channel analysis
6. Conclusion

Conclusion

- Lapin is linear $\rightarrow$ straightforward to mask
- Masked Lapin was implemented in an FPGA
  - Compact and very fast
  - Flexible datapath size (8-,16-,32-,64- and 128-bit)
  - High-order masking overhead increases linearly (quadratically for AES)
- Unprotected Lapin security to SCA was analyzed
  - Hamming weight model of accumulator
  - Attack based on prediction of t MSb and p LSb of s
  - Attacks exploiting reduction circuitry

Work in progress! **Thank you for attention!**

Extra slides

- Impl. issue: how to generate error bits with $\tau = 1/6 = 0.1\overline{6}$
- Close probabilities:

3-bit:	$1/8 = 0.125$,	$\Delta = -0.41\overline{6}$
4-bit:	$3/16 = 0.1875$,	$\Delta = +0.0208\overline{3}$
5-bit:	$5/32 = 0.15625$,	$\Delta = -0.01041\overline{6}$
6-bit:	$11/64 = 0.171875$,	$\Delta = +0.005208\overline{3}$
7-bit:	$21/128 = 0.1640625$,	$\Delta = -0.0026041\overline{6}$
8-bit:	$43/256 = 0.16796875$,	$\Delta = +0.00130208\overline{3}$
9-bit:	$85/512 = 0.166015625$,	$\Delta = -0.00065104\overline{16}$
10-bit:	$171/1024 = 0.1669921875$,	$\Delta = +0.00032552083$