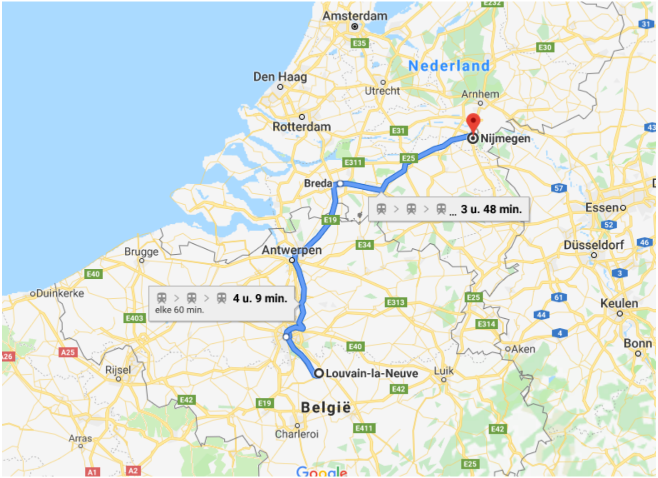
Vectorizing Higher-Order Masking

Ko Stoffelen

iCIS | Digital Security
Radboud University

Me


Amsterdam
Nederland
Den Haag
Utrecht
Arnhem
Rotterdam
Nijmegen
Breda
3 u. 48 min.
Essen
Düsseldorf
Antwerpen
Brugge
Duinkerke
4 u. 9 min.
elke 60 min.
Keulen
Bonn
Aken
Rijsel
Louvain-la-Neuve
Luik
België
Charleroi
Arras
E35
E30
E31
E311
E25
E19
E34
E40
E403
E313
E314
E42
E411
A25
A1
A2

Joint work

Based on a COSADE 2018 paper with the same title

Together with:
Benjamin Grégoire (INRIA Sophia Antipolis)
Kostas Papagiannopoulos (Radboud University)
Peter Schwabe (Radboud University)

Mandatory SCA slide

Boolean masking

- SCA countermeasure
- Use uniformly random value r to split secret variable x into uniformly random shares x_1 and x_2
- Set $x_1 := r$ and $x_2 := r \oplus x$; now $x_1 \oplus x_2 = x$
- Computations on x now on its shares
  - Easy for linear operations
  - Trickier for non-linear operations
- Computation becomes more expensive, but...
- Much (exponentially) harder for the attacker: needs to combine leakage of both shares to recover x
- Generalized to masking with d shares: $(d-1)$-order masking

Higher-order masking in practice

Higher-order masking is slow

Compare plot [GR17] to unmasked AES on somewhat similar CPU architecture: ~640 cycles

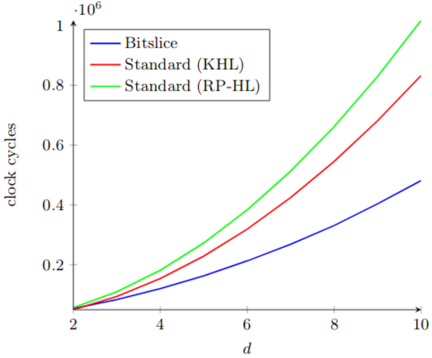


Fig. 20. Timings of masked AES.

Core ideas

- Use parallelism to improve efficiency of higher-order masking
- Use NEON vector registers on ARM Cortex-A8 for optimized 4-share and 8-share bitsliced AES
- Benchmark and evaluate its security against side-channel analysis

Bitslicing and AES

- Software implementation technique to easily operate on individual bits
- "Mimic hardware in software"
- Traditional bitslicing: store all bits in separate CPU registers
- E.g., for AES: 128 registers that each contain 1 bit
- If register has width w, process w independent blocks in parallel to improve throughput
- Disadvantage: you do not have 128 registers
- Disadvantage: you may not have w parallelizable blocks
- Instead: exploit internal parallelism of SubBytes in AES (or other SPN cipher)
- Store *every i'th bit of all state bytes* in separate CPU registers
- For AES: 8 registers that each contain 16 bits
- Process $\lfloor \frac{w}{16} \rfloor$ blocks in parallel

ARM CPUs

- Cortex-A (application): smartphone/tablet main CPU
- Cortex-R (real-time): sensors, PLCs, automotive
- Cortex-M (microcontroller): embedded controllers, IoT
- Our target: Cortex-A8
- 32-bit ARMv7-A architecture
- Comes with NEON unit for *Advanced SIMD* extension
- Adds vector registers and instructions

Masked bitsliced AES with NEON

- $16\times$ 128-bit register *or* $32\times$ 64-bit register
- Process *shares* in parallel instead of *independent blocks*

4 shares, 1 block

x_1	x_2	x_3	x_4

8 shares, 1 block

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

4 shares, 2 blocks

x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4

Parallel masking

- Problem: probing model unsuited for parallel implementations
- EUROCRYPT 2017: bounded moment model [BDF+17]
- Implementation is secure at order o if all mixed statistical moments of order $\leq o$ are independent of secret
- Serial security in probing model implies parallel security in bounded moment model
- Formal methods can be used to prove these properties
- So what kind of algorithms are secure in this model?

Secure parallel computations

These operations are sufficient

Addition/XOR

Simple: `veor` instruction

Multiplication/AND

Tricky: shares have to be combined to compute all partial products, but without leaking; requires fresh randomness

Refreshing

Use fresh randomness to re-create uniform distribution

Secure parallel refreshing/multiplication

- Gadgets should be *composable*
- Composability requires strong non-interference (SNI) [BBD+16]
- Use program verification to prove SNI and security in probing model
- This implies security in bounded moment model
- We could improve some earlier results, but results are hard to generalize

SNI-secure parallel refreshing

Notation: $\mathbf{x} = [x_1, \ldots, x_d]$; $\mathrm{rot}(\mathbf{x}, n) = [x_{1+n}, \ldots, x_d, x_1, \ldots, x_n]$

4 shares

$$\mathbf{r} \oplus \mathrm{rot}(\mathbf{r}, 1) \oplus \mathbf{x}$$

8 shares

Was

$$\mathbf{r} \oplus \mathrm{rot}(\mathbf{r}, 1) \oplus \mathbf{r}' \oplus \mathrm{rot}(\mathbf{r}', 1) \oplus \mathbf{r}'' \oplus \mathrm{rot}(\mathbf{r}'', 1) \oplus \mathbf{x}$$

Now

$$\mathbf{r} \oplus \mathrm{rot}(\mathbf{r}, 1) \oplus \mathbf{r}' \oplus \mathrm{rot}(\mathbf{r}', 2) \oplus \mathbf{x}$$

SNI-secure parallel refreshing

4 shares

```
vld1.64 {\tmp}, [\rand]!
veor \a, \tmp
vext.16 \tmp, \tmp, #1
veor \a, \tmp
```

8 shares

```
vld1.64 {\tmp}, [\rand:128]!
veor \a, \tmp
vext.16 \tmp, \tmp, #1
veor \a, \tmp

vld1.64 {\tmp}, [\rand:128]!
veor \a, \tmp
vext.16 \tmp, \tmp, #2
veor \a, \tmp
```

SNI-secure parallel multiplication

4 shares

Was

$$\mathbf{x} \cdot \mathbf{y} \oplus \mathbf{r} \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 1) \oplus \mathrm{rot}(\mathbf{x}, 1) \cdot \mathbf{y} \\ \oplus \mathrm{rot}(\mathbf{r}, 1) \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 2) \oplus \mathbf{r}' \oplus \mathrm{rot}(\mathbf{r}', 1)$$

Now

$$\mathbf{x} \cdot \mathbf{y} \oplus \mathbf{r} \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 1) \oplus \mathrm{rot}(\mathbf{x}, 1) \cdot \mathbf{y} \\ \oplus \mathrm{rot}(\mathbf{r}, 1) \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 2) \oplus [r', r', r', r']$$

8 shares

$$\mathbf{x} \cdot \mathbf{y} \oplus \mathbf{r} \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 1) \oplus \mathrm{rot}(\mathbf{x}, 1) \cdot \mathbf{y} \oplus \mathrm{rot}(\mathbf{r}, 1) \\ \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 2) \oplus \mathrm{rot}(\mathbf{x}, 2) \cdot \mathbf{y} \oplus \mathbf{r}' \\ \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 3) \oplus \mathrm{rot}(\mathbf{x}, 3) \cdot \mathbf{y} \oplus \mathrm{rot}(\mathbf{r}', 1) \\ \oplus \mathbf{x} \cdot \mathrm{rot}(\mathbf{y}, 4) \oplus \mathbf{r}'' \oplus \mathrm{rot}(\mathbf{r}'', 1)$$

SNI-secure parallel multiplication

4 shares

```
vand \c, \a, \b //K = A.B
vld1.64 {\tmpr}, [\rand]! //get 8 bytes of randomness
vext.16 \tmp, \b, \b, #1
veor \c, \tmpr  // + R
vand \tmp, \a
veor \c, \tmp   // + A.(rot B 1)
vext.16 \tmp, \a, \a, #1
vand \tmp, \b
veor \c, \tmp   // + (rot A 1).B
vext.16 \tmpr, \tmpr, #1
veor \c, \tmpr  // + (rot R 1)
vext.16 \tmp, \b, \b, #2
vand \tmp, \a
veor \c, \tmp   // + A.(rot B 2)
vld1.16 {\tmp[]}, [\rand]! //get 2 bytes of randomness
veor \c, \tmp   // + (r',r',r',r')
```

AES – SubBytes

- Circuit with least operations requires 81 XORs and 32 ANDs
- Use compiler from [BBD+16] to generate masked implementation with new gadgets
- Compiler detects when refreshing is necessary
- In this case: one input of every AND is refreshed
- Tool-assisted optimization: reschedule to decrease number of loads/stores
- Manual optimization: hide some CPU latencies, handle alignment issues

AES – ShiftRows

- Normal representation: rotation of rows
- Bitsliced representation: for all registers, for all shares, rotation within every 4 bits of the 16 bits

```
state[i] = ((state[i] & 0xf000) |
      ((state[i] & 0x0800) >> 3) |
      ((state[i] & 0x0700) << 1) |
      ((state[i] & 0x0030) << 2) |
      ((state[i] & 0x00c0) >> 2) |
      ((state[i] & 0x000e) >> 1) |
      ((state[i] & 0x0001) << 3)
);
```

Assembly: `vand`, `vmov.I16`, `vorr`, `vshl.I16`, `vsra.U16`

AES – MixColumns

- Normal representation: ‘matrix multiplication’ on columns
- Bitsliced representation: many XORs and rotations by multiples of 4 over 16 bits
- Assembly: `veor`, `vmov`, `vrev16.8`, `vshl.I16`, `vsra.U16`

AES – Randomness (bytes)

	4 shares	8 shares
Refreshing	8	32 (was 48)
Multiplication	10 (was 16)	48
Full AES	5,760	25,600

Speed of RNG has large impact on performance!

AES – Performance on Cortex-A8

	4 shares 1 block	4 shares 2 blocks	8 shares 1 block
Clock cycles (rand. from `/dev/urandom`)	1,598,133	4,738,024	9,470,743
Clock cycles (rand. from normal file)	14,488	17,586	26,601
Clock cycles (pre-loaded rand.)	12,385/ 774 cpb	15,194/ 475 cpb	23,616/ 1476 cpb
Stack usage in bytes	12	300	300
Code size in bytes	39,748	44,004	70,188

AES – Performance on Cortex-A8

[GR17] assumes pre-loaded randomness

From 100k/350k to 12k/24k

But Cortex-A8 more powerful

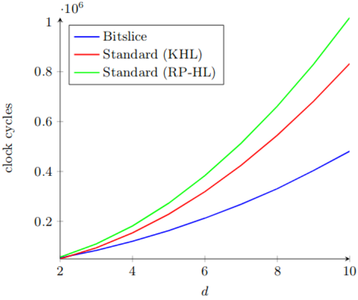
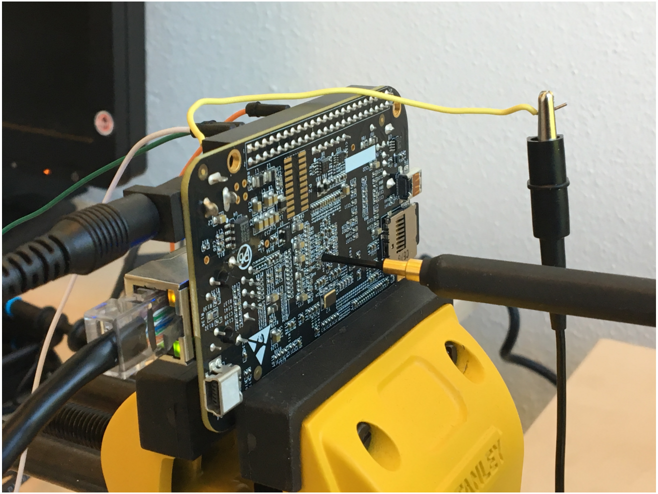


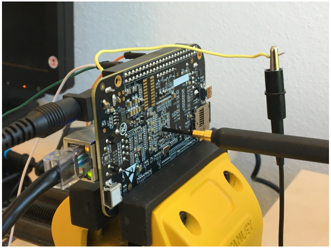
Fig. 20. Timings of masked AES.

SCA evaluation setup



SCA evaluation setup

- BeagleBone Black @ 1 GHz, running Debian
- LeCroy WaveRunner @ 2.5 GS/s for 1M traces
- Langer EM probe RF-B 0.3-3 @ capacitor 66
- Langer amplifier PA 303 SMA
- Trigger using GPIO port
- Data over Ethernet/TCP
- Elastic alignment post-processing



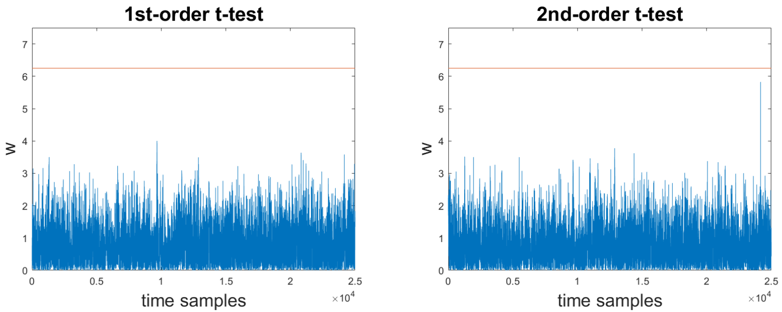
Share independence

- Ideally, d-share schemes are secure against $(d-1)$-order attacks
- Share recombination, coupling effects, distance-based leakage cause divergence
- We do not explicitly take care of these transitional leakages
- Practical security order $< d-1$
- Order reduction theorem: practical security order $\left\lfloor \frac{d-1}{2} \right\rfloor$ [BGG+14]
- So when $d = 4$, 1st-order security?

TVLA

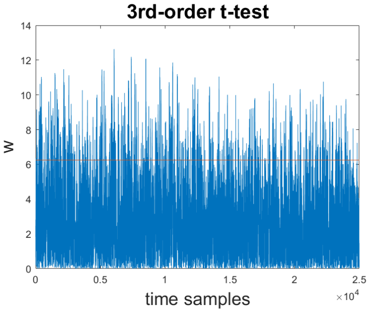
- First approach: Welch T-test
- Univariate 1M fixed vs. 1M random
- To keep computation time somewhat reasonable: focus on one AES round
- Use one-pass formulas of Schneider and Moradi [SM15]
- Many samples per trace: control familywise error rate with Ŝidak correction
- For 25k samples, threshold 6.25

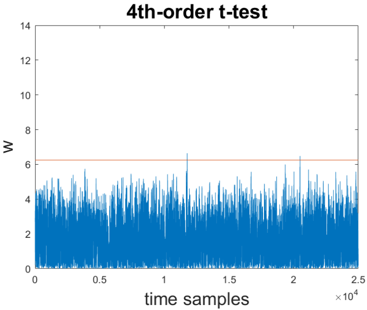
TVLA



T-test suggests resistance against 2nd-order attacks

TVLA

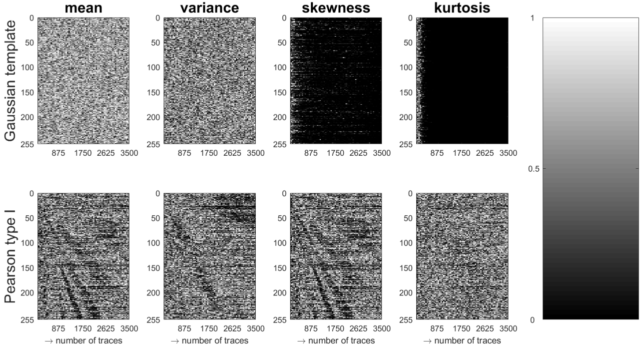




Security issues at 3rd order

Leakage certification

- Two types of errors [DSDP16]
  - Estimation errors: not enough traces
  - Modelling errors: incorrect leakage assumption
- Leakage certification can distinguish between them



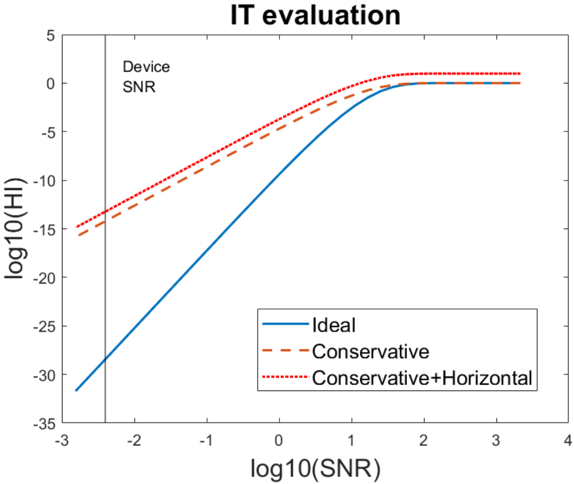
Information-theoretic bounds

- The previous approaches scale poorly to our 8-share implementation
- How to evaluate this? [DFS15]
  1. Estimate the SNR of the device (≈ 0.004)
  2. Compute the hypothetical information between the leakage and the secret key

     $$HI(S;L) = H[S] + \sum_{s \in S} \Pr[s] \cdot \int_{\ell \in \mathcal{L}} \widehat{\Pr}[\ell|s] \cdot \log_2 \widehat{\Pr}_{\mathrm{model}}[s|\ell] \mathrm{d}\ell$$

     This shows the 'amount' of leakage if estimated $\widehat{\Pr}_{\mathrm{model}}$ is accurate
  3. Extrapolate to 8 shares using information-theoretical bounds
- We use Prouff–Rivain bound: $1.72d + 2.72$ [PR13]

Information-theoretic bounds



Conclusions

- ARM NEON is a powerful tool for implementors
- Parallellized implementations become increasingly relevant in the context of SCA countermeasures
- Ensuring share independence seems to be hard and interfaces with the architectural and electrical layers
- Understanding the randomness requirements for masking / an efficient masking RNG is still an important open problem

Thanks. . .

. . . for your attention!

Questions?

References I

Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini.
Strong non-interference and type-directed higher-order masking.
In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 116–129. ACM, 2016.
http://eprint.iacr.org/2015/506.pdf.

Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub.
Parallel implementations of masking schemes and the bounded moment leakage model.
In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology — EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 535–566. Springer, 2017.
http://eprint.iacr.org/2016/912.pdf.

Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert.
On the cost of lazy engineering for masked software implementations.
In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications — CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer, 2014.
http://eprint.iacr.org/2014/413.pdf.

References II

Alexandre Duc, Sebastian Faust, and François-Xavier Standaert.
Making masking security proofs concrete — Or how to evaluate the security of any leaking device.
In *Advances in Cryptology — EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 401–429. Springer, 2015.
https://eprint.iacr.org/2015/119.pdf.

François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo.
Towards easy leakage certification.
In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems — CHES 2016*, volume 9813 of *LNCS*, pages 40–60. Springer, 2016.
https://eprint.iacr.org/2015/537.pdf.

Dahmun Goudarzi and Matthieu Rivain.
How fast can higher-order masking be in software?
In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology — EUROCRYPT 2017*, volume 10210 of *LNCS*, pages 567–597. Springer, 2017.
https://eprint.iacr.org/2016/264.pdf.

References III

Emmanuel Prouff and Matthieu Rivain.
Masking against side-channel attacks: A formal security proof.
In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.
http://www.iacr.org/archive/eurocrypt2013/78810139/78810139.pdf.

Tobias Schneider and Amir Moradi.
Leakage assessment methodology — A clear roadmap for side-channel evaluations.
In *Cryptographic Hardware and Embedded Systems — CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, 2015.
http://eprint.iacr.org/2015/207.pdf.