Secure and Efficient Masking of Lightweight Ciphers in Software and Hardware

Olivier Bronchain  Gaëtan Cassiers  François-Xavier Standaert

LWC NIST Workshop 2020
Introduction

Masking overview

Security vs Performance Analysis

First step: comparison proxies

Conclusion
Side-Channel Security at the Mode Level

Integrity and confidentiality at the mode level with side-channel:

- Requires different protection levels for parts of an AEAD [Bel+20b].
- Some need DPA (many inputs attack) protection everywhere.
- Some allow a mix of DPA / SPA (few inputs) security and unbounded leakage.

Examples for integrity (qualitatively):

OCB-Pyjamask

Spook
How to Reach DPA Security?

DPA security is required in many LWC candidates:
- Reach it by reducing DPA security to averaged-SPA security:
  - Isap and DryGascon
- Reach it through the use of masking:
  - Ascon, Spook, OCB-Pyjamask, ...

  Other implementation-level DPA countermeasures: less studied, part of this talk still applies.

In this talk we focus masking since it is well suited for many schemes:
1. How to implement safely and efficiently in software and hardware?
2. How to compare candidates w.r.t. masking & SCA protections?
Content

Introduction

Masking overview

Security vs Performance Analysis

First step: comparison proxies

Conclusion
Masking: general principles

Idea: share variables and replace logic gates with “gadgets”.

\[ x = x_1 \oplus x_2 \oplus \cdots \oplus x_d \]

Masking enables “t-probing secure” implementations [ISW03].

Cost of secure gadgets:

- linear: \( \mathcal{O}(d) \) (e.g. XOR gate)
- non-linear: \( \mathcal{O}(d^2) \) (e.g. AND gate)
- refresh: \( \mathcal{O}(d \log d) \) (sometimes required for secure composition)

Robust probing model for physical “imperfections” (i.e. glitches, transitions)
A Brief Timeline of Software Masking

Over the last decade:

CHES10 [RP10]
   Implementation of [ISW03] on MCU.
FSE13 [Cor+13]
   Attack on [RP10]: composition issue due to weak refreshing.
Eurocrypt17 [GR17]
   Efficient bitslice masking (proven secure in [CS20]).
Asiacrypt18 [BGR18]
   \textit{Tight private circuits (TPC)}: improved efficiency (probing secure).
Eurocrypt20 [Bel+20a]
   \textit{Tornado}: TPC with register-probing security & automated code generation.
A Brief Timeline of Hardware Masking

Some *glitch-robust probing* secure schemes from the last decade:

**TI [NRS11]**

*Non-completeness + uniformity* ⇒ first-order glitch-robust probing secure.


Higher-order glitch-robust optimized AND gadgets.

**[Moo+19]**

Probing attacks against CMS/DOM/UMA/…

**HPC [Cas+20]**

Provably secure AND gadgets & *fullVerif* composition verification tool.
Content

Introduction

Masking overview

Security vs Performance Analysis

First step: comparison proxies

Conclusion
How to Compare Candidates?

It should go in 3 steps:
1. Implement
2. Evaluate performance
3. Evaluate side-channel security

Challenges:
- Evaluate algorithms and not the masking schemes
  - Many optimized implementations for each candidate
- Accurate security evaluation

Given limited expert bandwidth
Side-Channel Security Evaluation

- Probing security verification
  - Algorithmic security order reductions

- Robust probing security verification
  - Alg. and some physical order reductions

- Test Vector Leakage Assessment (TVLA)
  - Detects order
  - Based on measurements
  - Limited to low order, low dimensionality verification
  - Risk of false negative

- Best attack
  - Can spot multiple kinds of weaknesses
  - Highly time consuming and skills required (e.g. Spook CTF)
Content

Introduction

Masking overview

Security vs Performance Analysis

First step: comparison proxies

Conclusion
Proxy 1: Count masked AND gates

Starting point:

*Masked AND gates make most of the cost of (high-order) implementations.*

Software Implementation:
- Clock cycles
- Required randomness
- ...

Hardware Implementation:
- Latency
- Required randomness
- Area
- ...

Limitations:
- Ignores the rest of the computation (not free!)
- Structure of the cipher also has an impact (e.g. depth)

Integrate counts from [Mey20] with mode-level requirements.
Proxy 1: AND gates per encrypted byte

For integrity (CIML2) only!
Proxy 2: Tornado

Tornado:
- Automated masked C code generation.
- +/-30% overhead w.r.t. hand-optimized.
- Ensure register-probing security.
- TPC+ masking scheme.

⇒ Tornado implementations hardly comparable to hand-optimized ones.

Not a magic tool:
- worst-case security (e.g. transitions)?
- optimal performance?
- other masking schemes?

Suggestion: Compare Tornado implementations of candidates
- More realistic than counting masked AND gates
- Easy/Fast implementation: high-level description of primitive
  - 11 candidate’s primitives already done by the authors of Tornado
Content

Introduction

Masking overview

Security vs Performance Analysis

First step: comparison proxies

Conclusion
Conclusion

Approaches to compare SCA robustness of candidates:

- **Best implementations and best attacks:**
  - *Both implementing and evaluating require expertise and time.*
  - *May evaluate the implementer’s skills more than the candidates.*
  - Useful byproduct: good implementation of the winner(s)?

- **Proxies:**
  - Counting masked AND gates,
  - Tornado: automated software masking,
  - Others?

Our opinion

- Proxies are more relevant than best implementation & attacks, esp. given resource constraints.
- The proposed proxies already have a good comparative value.
References I


References II


References III


