Optimal First-Order Boolean Masking for Embeded IoT Devices

Alex Biryukov, Daniel Dinu, Yann Le Corre, Aleksei Udovenko

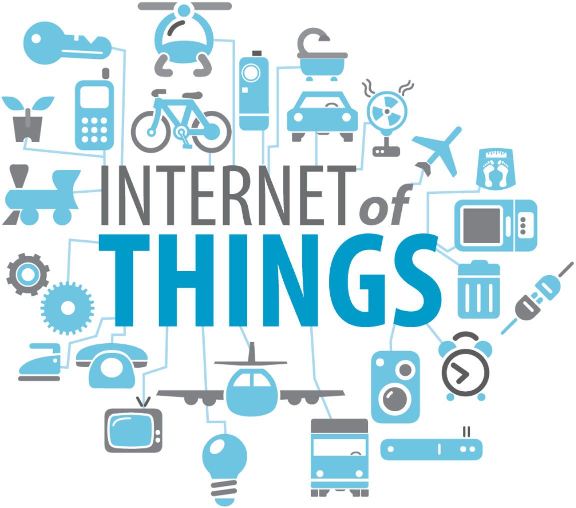
University of Luxembourg, SnT

November 13, 2017

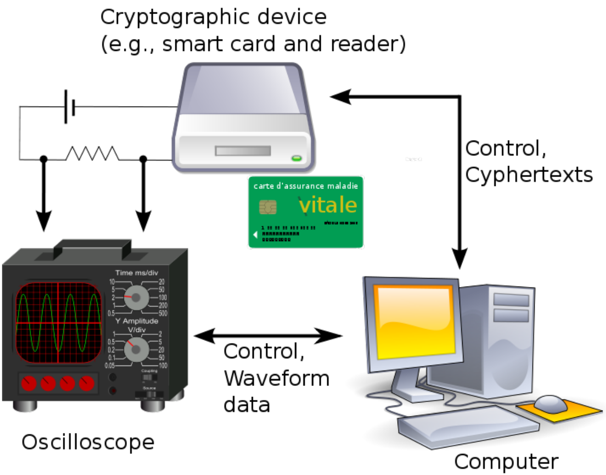
Outline

Plan

Internet of Things



Side Channel Attacks



[1] Credit: wikipedia

Masking (1/3)

Countermeasure - masking (first-order example):

- Represent $x \sim (r_x, x')$ such that $x = r_x \oplus x'$.
  - r_x is a random bit,
  - $x' = x \oplus r_x$.

Masking (1/3)

Countermeasure - masking (first-order example):

- Represent $x \sim (r_x, x')$ such that $x = r_x \oplus x'$.
  - r_x is a random bit,
  - $x' = x \oplus r_x$.
- Compute using new representation without leaking x.

Masking (1/3)

Countermeasure - masking (first-order example):

- Represent $x \sim (r_x, x')$ such that $x = r_x \oplus x'$.
  - r_x is a random bit,
  - $x' = x \oplus r_x$.
- Compute using new representation without leaking x.
- **Example 1** (XOR):

$$\begin{aligned} x &\sim (r_x, x'), \\ y &\sim (r_y, y'), \\ x \oplus y &\sim (r_x \oplus r_y, x' \oplus y'). \end{aligned}$$

Masking (1/3)

Countermeasure - masking (first-order example):

- Represent $x \sim (r_x, x')$ such that $x = r_x \oplus x'$.
  - r_x is a random bit,
  - $x' = x \oplus r_x$.
- Compute using new representation without leaking x.
- **Example 1** (XOR):

$$x \sim (r_x, x'),$$
$$y \sim (r_y, y'),$$
$$x \oplus y \sim (r_x \oplus r_y, x' \oplus y').$$

- **Example 2** (AND - Trichina gate):

$$x \wedge y \sim \big(r_z, r_z \oplus (r_x \wedge r_y) \oplus (r_x \wedge y) \oplus (x \wedge r_y) \oplus (x \wedge y)\big).$$

Masking (2/3)

- First-order masking can be broken using second-order attack.
- More shares $n \rightarrow$ higher-order masking.

Masking (2/3)

- First-order masking can be broken using second-order attack.
- More shares $n \rightarrow$ higher-order masking.
- Masking AND requires $O(n^2)$ operations.
- Attack complexity (data and time) grows exponentially.

Masking (2/3)

- First-order masking can be broken using second-order attack.
- More shares $n \rightarrow$ higher-order masking.
- Masking AND requires $O(n^2)$ operations.
- Attack complexity (data and time) grows exponentially.
- Some devices can not afford higher-order masking!
- Some protection is still desirable.

Masking (2/3)

- First-order masking can be broken using second-order attack.
- More shares $n \rightarrow$ higher-order masking.
- Masking AND requires $O(n^2)$ operations.
- Attack complexity (data and time) grows exponentially.
- Some devices can not afford higher-order masking!
- Some protection is still desirable.
- $\Rightarrow$ Efficient first-order masking is necessary.

Masking (3/3)

Best known first-order expressions?

Masking (3/3)

Best known first-order expressions?

- **AND:** Trichina gate. 1 random bit and 8 basic operations.
- **OR:** Not studied? Using De Morgan's law and Trichina gate: 1 random bit and 11 basic operations.

Masking (3/3)

Best known first-order expressions?

- **AND:** Trichina gate. 1 random bit and 8 basic operations.
- **OR:** 6 basic operations, [Baek and Noh, 2005]

Masking (3/3)

Best known first-order expressions?

- **AND:** Trichina gate. 1 random bit and 8 basic operations.
- **OR:** 6 basic operations, [Baek and Noh, 2005]

1 fresh random bit required:

- [+]: masks are always "fresh" $\rightarrow$ easy security proof.
- [$-$]: PRNG cost.

Our goal: find optimal expressions, without randomness if possible.

Plan

Interface

Algorithm: search for optimal first-order masking expressions.

Interface

Algorithm: search for optimal first-order masking expressions.

Inputs:

- target Boolean function t. For example, AND:

$$t(x_0, x_1, y_0, y_1) = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1);$$

- number of output shares m;
- set of sensitive functions, e.g. $\{x_0 \oplus x_1, y_0 \oplus y_1, t\}$;
- set of allowed operations, e.g. $\{XOR, AND, OR, \underbrace{BIC, ORN}_{ARM-specific}\}$.

Interface

Algorithm: search for optimal first-order masking expressions.

Inputs:

- target Boolean function t. For example, AND:

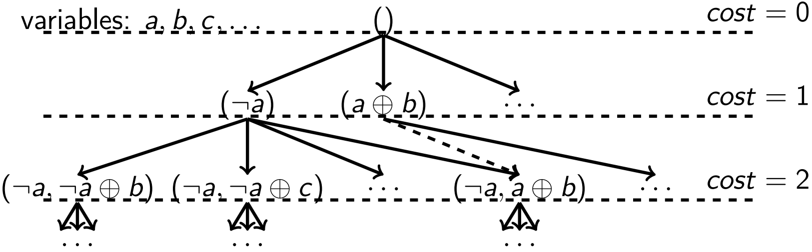
$$t(x_0, x_1, y_0, y_1) = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1);$$

- number of output shares m;
- set of sensitive functions, e.g. $\{x_0 \oplus x_1, y_0 \oplus y_1, t\}$;
- set of allowed operations, e.g. $\{XOR, AND, OR, \underbrace{BIC, ORN}_{ARM-specific}\}$.

Outputs:

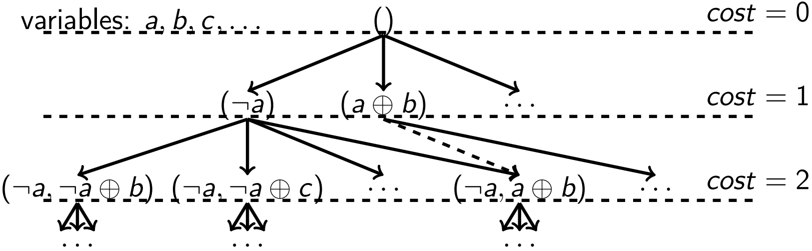
- set of m functions s_i such that $\bigoplus_i s_i = t$;
- optimal circuit for computing all s_i without first-order leakage of information about sensitive functions.

The Algorithm (1/3)



- A **breadth-first** search on sequences of operations.
- A sequence is good if it contains m functions summing to t.
- Several cut-offs involved.

The Algorithm (2/3)



Cut-offs:

- First-order leakage check. Leaking sequences are dropped.
- Two sequences with the same set of functions are merged.
- Exploiting share symmetries (swaps, etc.).

The Algorithm (3/3)

Example of a discovered sequence:

$\neg y_0$,
$x_0 \vee \neg y_1$,
$x_0 \wedge y_0$,
$(x_0 \wedge y_0) \oplus (x_0 \vee \neg y_1)$,
$x_1 \vee \neg y_1$,
$x_1 \wedge y_0$,
$(x_1 \wedge y_0) \oplus (x_1 \vee \neg y_1)$.

The Algorithm (3/3)

Example of a discovered sequence:

$\neg y_0$,
$x_0 \vee \neg y_1$,
$x_0 \wedge y_0$,
$(x_0 \wedge y_0) \oplus (x_0 \vee \neg y_1)$,
$x_1 \vee \neg y_1$,
$x_1 \wedge y_0$,
$(x_1 \wedge y_0) \oplus (x_1 \vee \neg y_1)$.

Observe that the sequence contains
$s_0 = (x_0 \wedge y_0) \oplus (x_0 \vee \neg y_1)$,
$s_1 = (x_1 \wedge y_0) \oplus (x_1 \vee \neg y_1)$,
such that
$s_0 \oplus s_1 = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1) = t$ is the target AND function.

Results

SecAnd (secure AND):
$z_0 = (x_1 \wedge y_1) \oplus (x_1 \vee \neg y_2)$,
$z_1 = (x_2 \wedge y_1) \oplus (x_2 \vee \neg y_2)$,
Cost: 7 basic / 6 on ARM (versus 8 Trichina gate).

SecOr (secure OR):
$z_0 = (x_1 \wedge y_1) \oplus (x_1 \vee y_2)$,
$z_1 = (x_2 \vee y_1) \oplus (x_2 \wedge y_2)$,
Cost: 6 basic / 6 on ARM (versus 11 Trichina gate + De Morgan's law).

Results

SecAnd (secure AND):
$z_0 = (x_1 \wedge y_1) \oplus (x_1 \vee \neg y_2)$,
$z_1 = (x_2 \wedge y_1) \oplus (x_2 \vee \neg y_2)$,
Cost: 7 basic / 6 on ARM (versus 8 Trichina gate).

SecOr (secure OR):
$z_0 = (x_1 \wedge y_1) \oplus (x_1 \vee y_2)$,
$z_1 = (x_2 \vee y_1) \oplus (x_2 \wedge y_2)$,
Cost: 6 basic / 6 on ARM (versus 11 Trichina gate + De Morgan's law).

No random bits required!

Plan

Applications

We applied new masking expressions to improve several algorithms:

- Masked Modular Addition/Subtraction by Coron *et al.* from FSE 2013.
- Masked top 3 64-bit block ciphers in the FELICS benchmarking framework:
  - Speck
  - Simon
  - Rectangle

Applications

We applied new masking expressions to improve several algorithms:

- Masked Modular Addition/Subtraction by Coron *et al.* from FSE 2013.
- Masked top 3 64-bit block ciphers in the FELICS benchmarking framework:
  - Speck
  - Simon
  - Rectangle

- All implementations were checked using Welch's t-test to verify absence of leakage (using simulated traces).
- Just a proof-of-concept to compare performance.
- More work is needed for deployment-ready implementations.

Kogge-Stone Addition/Subtraction

- Coron *et al.* at FSE 2013 proposed masked modular addition algorithm based on the Kogge-Stone adder.
- We used our new expressions together with other modifications.

Expr.	Time (cycles)		Code size (bytes)	
	Addition	Subtraction	Addition	Subtraction
		rolled		
best known	275	388	292	416
our	228	333	232	332
gain	**17%**	**14%**	**21%**	**20%**
		unrolled		
best known	203	296	544	812
our	173	241	480	692
gain	**15%**	**19%**	**12%**	**15%**

Speck

- **Speck:** ARX block cipher from NSA.
- **Speck-64/128:** 64-bit block, 128-bit key, 27 rounds.

Expr.	Time (cycles)		Code size (bytes)	
	Enc	Dec	Enc	Dec
rolled adder				
best known	7131	11368	340	488
our	5686	8258	272	400
gain	**21%**	**27%**	**20%**	**18%**
unrolled adder				
best known	4945	7431	588	876
our	4666	6188	536	712
gain	**6%**	**17%**	**9%**	**19%**

Simon

- **Simon:** AndRX block cipher from NSA.
- **Simon-64/128:** 64-bit block, 128-bit key, 44 rounds.

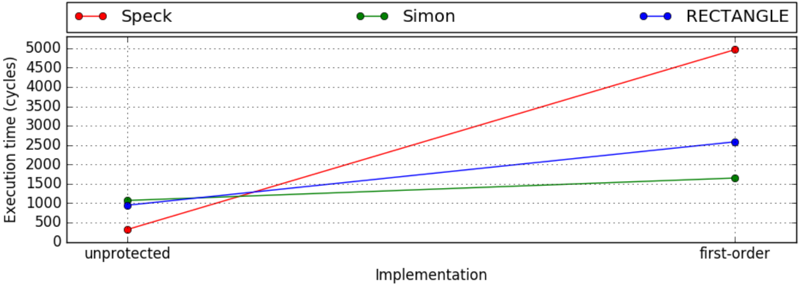
Expr.	Time (cycles)		Code size (bytes)	
	Enc	Dec	Enc	Dec
best known	1736	1737	152	156
our	1648	1649	136	140
gain	**5%**	**5%**	**27%**	**25%**

Rectangle

- **RECTANGLE:** bit-sliced block cipher from academia (Zhang *et al.*).
- **RECTANGLE-64/128:** 64-bit block, 128-bit key, 25 rounds.

Expr.	Time (cycles)		Code size (bytes)	
	Enc	Dec	Enc	Dec
best known	3661	3442	632	444
our	2584	2954	564	372
gain	**19%**	**14%**	**11%**	**16%**

First-Order Masking Penalty



Plan

Compositional Security (1/3)

Is it always secure to compose our **SecAnd** / **SecOr** operations?

Compositional Security (1/3)

Is it always secure to compose our **SecAnd** / **SecOr** operations?

Unfortunately, no! A simple counterexample by a reviewer:

$$(x \vee y) \wedge y.$$

Using our expressions to mask this circuit results in a first-order leakage.
Problem: dependent input masks to **SecAnd**.

Compositional Security (1/3)

Is it always secure to compose our **SecAnd** / **SecOr** operations?

Unfortunately, no! A simple counterexample by a reviewer:

$$(x \vee y) \wedge y.$$

Using our expressions to mask this circuit results in a first-order leakage.
Problem: dependent input masks to **SecAnd**.
Solution: ... remask! But not after each operation.

Compositional Security (2/3)

How often to remask?

Compositional Security (2/3)

How often to remask?

Consider for example **SecAnd**:

$$(z', r_z) = SecAnd((x', r_x), (y', r_y)),$$

After simplification, we have:

$$z' = z \oplus r_x y \oplus r_y \oplus 1,$$
$$r_z = r_x y \oplus r_y \oplus 1.$$

Observe that r_z is *linear* in r_x and r_y. However, the expression depends on the secret variable y.
Similar proposition holds for **SecOr** as well.

Compositional Security (3/3)

- We can track the coefficient vector of each share through the circuit.
- For example:
  - Consider 4 random shares $r_0, \ldots, r_3$.
  - Consider the random mask: $r_0 \oplus xr_1 \oplus r_2$.
  - We represent it as $(1, ?, 1, 0)$.
- **SecAnd** / **SecOr** are secure if the input vectors are independent.
- If the known vector coefficients of the shares match, we remask the shares before the operation.
- Otherwise masks are guaranteed to be *independent*.
- Requires case-by-case study - future work.

Plan

Conclusion

- New, optimal expressions for first-order masking.
- Decrease penalty of protecting lightweight block ciphers.

Open problems:

- Optimal remasking frequency?

Thank you!