Provably Secure Higher-Order Masking of AES

Matthieu Rivain — CryptoExperts
Emmanuel Prouff — Oberthur

CHES 2010, Santa Barbara, Aug. 20th

Outline

1. Introduction
   - Higher-Order Masking
   - ISW Scheme (CRYPTO'03)
2. Our Scheme
   - Masking the S-box
   - Masking the Whole AES
   - Security
   - Implementation Results
3. Conclusion

Outline

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \perp x_1 \perp \cdots \perp x_d = x$$

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

- The *masks* ($i \geq 1$): $x_i \leftarrow \$$

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

- The *masks* ($i \geq 1$): $x_i \leftarrow \$$
- The *masked variable*: $x_0 \leftarrow x \oplus x_1 \oplus \cdots \oplus x_d$

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

- The *masks* ($i \geq 1$): $x_i \leftarrow \$$
- The *masked variable*: $x_0 \leftarrow x \oplus x_1 \oplus \cdots \oplus x_d$
- Note: equiv. $d+1$ out of $d+1$ secret sharing of x

Higher-Order Masking
Basic principle

- Every key-dependent variable x is shared into $d+1$ variables

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

- The *masks* ($i \geq 1$): $x_i \leftarrow \$$
- The *masked variable*: $x_0 \leftarrow x \oplus x_1 \oplus \cdots \oplus x_d$
- Note: equiv. $d+1$ out of $d+1$ secret sharing of x
- Computation carried out by processing the shares separately

Higher-Order Masking
Soundness

[Chari-Jutla-Rao-Rohatgi CRYPTO'99]

- Bit x masked $\mapsto x_0, x_1, \ldots, x_d$
- Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$

Higher-Order Masking
Soundness

[Chari-Jutla-Rao-Rohatgi CRYPTO'99]

- Bit x masked $\mapsto x_0, x_1, \ldots, x_d$
- Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$
- Number of leakage samples to distinguish $\big((L_i)_i | x = 0\big)$ from $\big((L_i)_i | x = 1\big)$:

$$q \geq O(1)\sigma^d$$

Higher-Order Masking
Soundness

[Chari-Jutla-Rao-Rohatgi CRYPTO'99]

- Bit x masked $\mapsto x_0, x_1, \ldots, x_d$
- Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$
- Number of leakage samples to distinguish $\big((L_i)_i | x = 0\big)$ from $\big((L_i)_i | x = 1\big)$:

$$q \geq O(1)\sigma^d$$

Higher-order masking is sound in the presence of noisy leakage!

Higher-Order Masking Schemes

Definition

A *dth-order masking scheme* for an encryption algorithm $c \leftarrow \mathcal{E}(m, k)$ is an algorithm

$$(c_0, c_1, \ldots, c_d) \leftarrow \mathcal{E}'\big((m_0, m_1, \ldots, m_d), (k_0, k_1, \ldots, k_d)\big)$$

Higher-Order Masking Schemes

Definition

A *dth-order masking scheme* for an encryption algorithm $c \leftarrow \mathcal{E}(m, k)$ is an algorithm

$$(c_0, c_1, \ldots, c_d) \leftarrow \mathcal{E}'\big((m_0, m_1, \ldots, m_d), (k_0, k_1, \ldots, k_d)\big)$$

- *completeness*: $\bigoplus_i m_i = m$ and $\bigoplus_i k_i = k$

$$\Rightarrow \bigoplus_i c_i = \mathcal{E}(m, k)$$

Higher-Order Masking Schemes

Definition

A *dth-order masking scheme* for an encryption algorithm $c \leftarrow \mathcal{E}(m, k)$ is an algorithm

$$(c_0, c_1, \ldots, c_d) \leftarrow \mathcal{E}'\big((m_0, m_1, \ldots, m_d), (k_0, k_1, \ldots, k_d)\big)$$

- *completeness*: $\bigoplus_i m_i = m$ and $\bigoplus_i k_i = k$

$$\Rightarrow \bigoplus_i c_i = \mathcal{E}(m, k)$$

- *security*: $\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d$:

$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

Higher-Order Masking Schemes

Definition

A *dth-order masking scheme* for an encryption algorithm $c \leftarrow \mathcal{E}(m, k)$ is an algorithm

$$(c_0, c_1, \ldots, c_d) \leftarrow \mathcal{E}'\big((m_0, m_1, \ldots, m_d), (k_0, k_1, \ldots, k_d)\big)$$

- *completeness*: $\bigoplus_i m_i = m$ and $\bigoplus_i k_i = k$

$$\Rightarrow \bigoplus_i c_i = \mathcal{E}(m, k)$$

- *security*: $\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d$:

$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

For SPN (*eg.* DES, AES) the main issue is masking the S-box.

Higher-Order Masking Schemes
Literature

Software implementations:

- [Schramm-Paar CT-RSA'06]
  - secure only for $d \leq 2$ [Coron-Prouff-Rivain CHES'07]

Higher-Order Masking Schemes
Literature

Software implementations:

- [Schramm-Paar CT-RSA'06]
  - secure only for $d \leq 2$ [Coron-Prouff-Rivain CHES'07]
- [Rivain-Dottax-Prouff FSE'08]
  - alternative solutions dedicated to $d = 2$

Higher-Order Masking Schemes
Literature

Software implementations:

- [Schramm-Paar CT-RSA'06]
  - secure only for $d \leq 2$ [Coron-Prouff-Rivain CHES'07]
- [Rivain-Dottax-Prouff FSE'08]
  - alternative solutions dedicated to $d = 2$

Hardware implementations:

- [Ishai-Sahai-Wagner CRYPTO'03]
  - every wire/logic gate is masked at an arbitrary order d
  - wires values $\equiv$ intermediate variables

    $\Rightarrow$ dth-order masking scheme

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = (\bigoplus_i a_i)(\bigoplus_i b_i) = \bigoplus_{i,j} a_i b_j$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = (\bigoplus_i a_i)(\bigoplus_i b_i) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 & a_0b_2 \\ a_1b_0 & a_1b_1 & a_1b_2 \\ a_2b_0 & a_2b_1 & a_2b_2 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 & a_0b_2 \\ 0 & a_1b_1 & a_1b_2 \\ 0 & 0 & a_2b_2 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ a_1b_0 & 0 & 0 \\ a_2b_0 & a_2b_1 & 0 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = (\bigoplus_i a_i)(\bigoplus_i b_i) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 & a_0b_2 \\ 0 & a_1b_1 & a_1b_2 \\ 0 & 0 & a_2b_2 \end{pmatrix} \oplus \begin{pmatrix} 0 & a_1b_0 & a_2b_0 \\ 0 & 0 & a_2b_1 \\ 0 & 0 & 0 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 \oplus a_1b_0 & a_0b_2 \oplus a_2b_0 \\ 0 & a_1b_1 & a_1b_2 \oplus a_2b_1 \\ 0 & 0 & a_2b_2 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 \oplus a_1b_0 & a_0b_2 \oplus a_2b_0 \\ 0 & a_1b_1 & a_1b_2 \oplus a_2b_1 \\ 0 & 0 & a_2b_2 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 \oplus a_1b_0 & a_0b_2 \oplus a_2b_0 \\ 0 & a_1b_1 & a_1b_2 \oplus a_2b_1 \\ 0 & 0 & a_2b_2 \end{pmatrix} \oplus \begin{pmatrix} 0 & r_{1,2} & r_{1,3} \\ 0 & 0 & r_{2,3} \\ 0 & 0 & 0 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & a_0b_1 \oplus a_1b_0 & a_0b_2 \oplus a_2b_0 \\ 0 & a_1b_1 & a_1b_2 \oplus a_2b_1 \\ 0 & 0 & a_2b_2 \end{pmatrix} \oplus \begin{pmatrix} 0 & r_{1,2} & r_{1,3} \\ r_{1,2} & 0 & r_{2,3} \\ r_{1,3} & r_{2,3} & 0 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & & \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

- Ishai *et al.* prove $(d/2)$th-order security

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

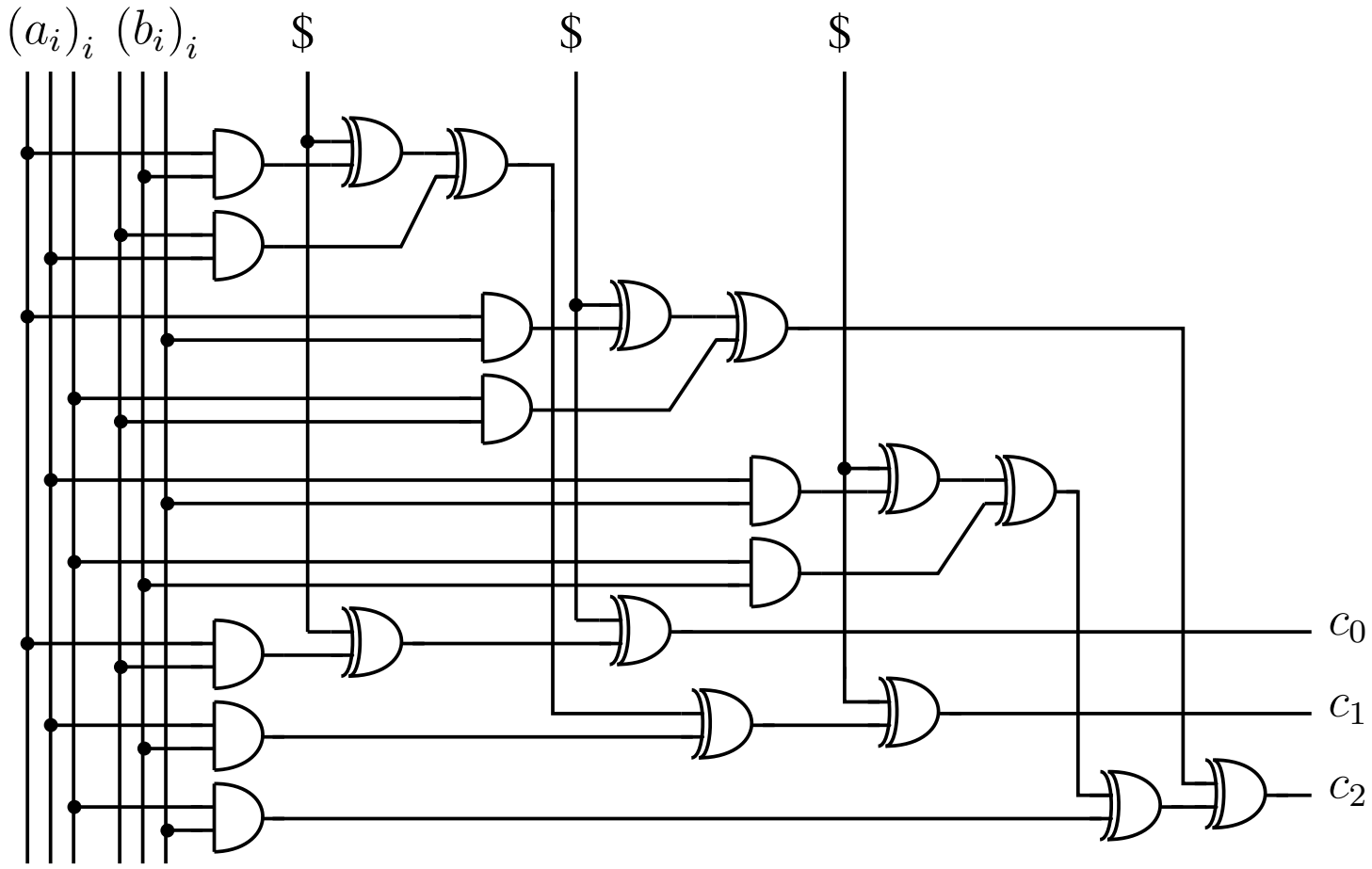
- Example ($d = 2$):

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

- Ishai *et al.* prove $(d/2)$th-order security
  - We prove dth-order security

Ishai-Sahai-Wagner (ISW) Scheme

Example: AND gate for $d = 2$



Ishai-Sahai-Wagner (ISW) Scheme

Practical Issues

- Important area overhead for the masked circuit
  - A wire is encoded by $d+1$ wires
  - One AND gate encoded by
    - $(d+1)^2$ ANDs $+$ $2d(d+1)$ XORs $+$ $d(d+1)/2$ \$

Ishai-Sahai-Wagner (ISW) Scheme

Practical Issues

- Important area overhead for the masked circuit
  - A wire is encoded by $d+1$ wires
  - One AND gate encoded by
    - $(d+1)^2$ ANDs + $2d(d+1)$ XORs + $d(d+1)/2$ \$
  - Example: AES S-box circuit

	ISW		
No masking	$d=1$	$d=2$	$d=3$
200 gates	500 gates	1.1 Kgates	2 Kgates

Ishai-Sahai-Wagner (ISW) Scheme

Practical Issues

- Important area overhead for the masked circuit
  - A wire is encoded by $d+1$ wires
  - One AND gate encoded by
    - $(d+1)^2$ ANDs + $2d(d+1)$ XORs + $d(d+1)/2$ \$
  - Example: AES S-box circuit

	ISW		
No masking	$d=1$	$d=2$	$d=3$
200 gates	500 gates	1.1 Kgates	2 Kgates

- Practical security issue with glitches
  - addition of synchronizing elements $\Rightarrow$ additional overhead

Ishai-Sahai-Wagner (ISW) Scheme

Practical Issues

- Important area overhead for the masked circuit
  - A wire is encoded by $d+1$ wires
  - One AND gate encoded by
    - $(d+1)^2$ ANDs + $2d(d+1)$ XORs + $d(d+1)/2$ \$
  - Example: AES S-box circuit

	ISW		
No masking	$d=1$	$d=2$	$d=3$
200 gates	500 gates	1.1 Kgates	2 Kgates

- Practical security issue with glitches
  - addition of synchronizing elements $\Rightarrow$ additional overhead
- Not suitable for software implementations

Outline

Masking the S-box

- Non-linearity $\Rightarrow$ difficulty to mask

Masking the S-box

- Non-linearity $\Rightarrow$ difficulty to mask
- We use the AES S-box structure: $\mathrm{S} = \mathrm{Exp} \circ \mathrm{Af}$
  - Af: affine transformation over $\mathbb{F}_2^8$
  - Exp : $x \mapsto x^{254}$ over $\mathbb{F}_{256}$

Masking the S-box

- Non-linearity $\Rightarrow$ difficulty to mask
- We use the AES S-box structure: $\mathsf{S} = \mathsf{Exp} \circ \mathsf{Af}$
  - Af: affine transformation over $\mathbb{F}_2^8$
  - Exp : $x \mapsto x^{254}$ over $\mathbb{F}_{256}$
- Masking Af is easy:

$$\mathsf{Af}(x) = \mathsf{Af}(x_0) \oplus \mathsf{Af}(x_1) \oplus \cdots \oplus \mathsf{Af}(x_d) \oplus \texttt{0x63} \quad \text{iff } d \text{ is odd}$$

Masking the S-box

- Non-linearity $\Rightarrow$ difficulty to mask
- We use the AES S-box structure: $\mathsf{S} = \mathsf{Exp} \circ \mathsf{Af}$
  - Af: affine transformation over $\mathbb{F}_2^8$
  - Exp : $x \mapsto x^{254}$ over $\mathbb{F}_{256}$
- Masking Af is easy:

$$\mathsf{Af}(x) = \mathsf{Af}(x_0) \oplus \mathsf{Af}(x_1) \oplus \cdots \oplus \mathsf{Af}(x_d) \oplus \texttt{0x63} \quad \text{iff } d \text{ is odd}$$

- For Exp we use an exponentiation algorithm
  - approach used for 1st-order masking in [Blömer-Merchan-Krummel SAC'04]
  - we want to design a dth-order secure exponentiation
  - we need dth-order secure square and multiplication

Masking the S-box

- dth-order secure square
  - squaring is linear over $\mathbb{F}_{256}$

$$x_0^2 \oplus x_1^2 \oplus \cdots \oplus x_d^2 = x^2$$

Masking the S-box

- dth-order secure square
  - squaring is linear over $\mathbb{F}_{256}$

$$x_0^{2^j} \oplus x_1^{2^j} \oplus \cdots \oplus x_d^{2^j} = x^{2^j}$$

Masking the S-box

- dth-order secure square
  - squaring is linear over $\mathbb{F}_{256}$

$$x_0^{2^j} \oplus x_1^{2^j} \oplus \cdots \oplus x_d^{2^j} = x^{2^j}$$

- dth-order secure multiplication
  - we generalize the ISW scheme to $\mathbb{F}_{256}$
    - AND $\Rightarrow$ $\mathbb{F}_{256}$ multiplication
    - XOR $\Rightarrow$ $\mathbb{F}_{256}$ addition (8-bit XOR)
    - $\$_1 \Rightarrow \$_8$ (random 8-bit value)

Masking the S-box

- dth-order secure square
  - squaring is linear over $\mathbb{F}_{256}$

$$x_0^{2^j} \oplus x_1^{2^j} \oplus \cdots \oplus x_d^{2^j} = x^{2^j}$$

- dth-order secure multiplication
  - we generalize the ISW scheme to $\mathbb{F}_{256}$
    - AND $\Rightarrow$ $\mathbb{F}_{256}$ multiplication
    - XOR $\Rightarrow$ $\mathbb{F}_{256}$ addition (8-bit XOR)
    - $\$_1 \Rightarrow \$_8$ (random 8-bit value)
- Complexity:
  - secure square: $d+1$ squares
  - secure mult: $(d+1)^2$ mult, $2d(d+1)$ XOR, $d(d+1)/2$ $\$_8$

Masking the S-box

- dth-order secure square
  - squaring is linear over $\mathbb{F}_{256}$

$$x_0^{2^j} \oplus x_1^{2^j} \oplus \cdots \oplus x_d^{2^j} = x^{2^j}$$

- dth-order secure multiplication
  - we generalize the ISW scheme to $\mathbb{F}_{256}$
    - AND $\Rightarrow$ $\mathbb{F}_{256}$ multiplication
    - XOR $\Rightarrow$ $\mathbb{F}_{256}$ addition (8-bit XOR)
    - $\$_1 \Rightarrow \$_8$ (random 8-bit value)
- Complexity:
  - secure square: $d+1$ squares
  - secure mult: $(d+1)^2$ mult, $2d(d+1)$ XOR, $d(d+1)/2$ $\$_8$
- Our goal: minimize the number of multiplications which are not squares

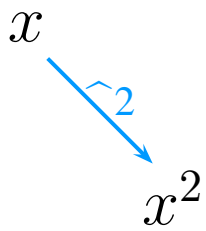
Masking the S-box

The proposed addition chain:

$$x$$

Masking the S-box

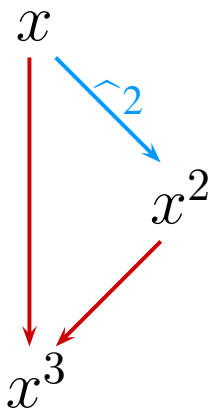
The proposed addition chain:



- one square

Masking the S-box

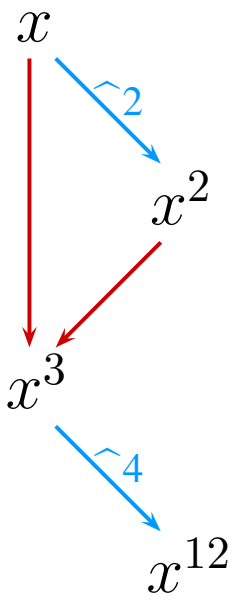
The proposed addition chain:



- one square
- one mult

Masking the S-box

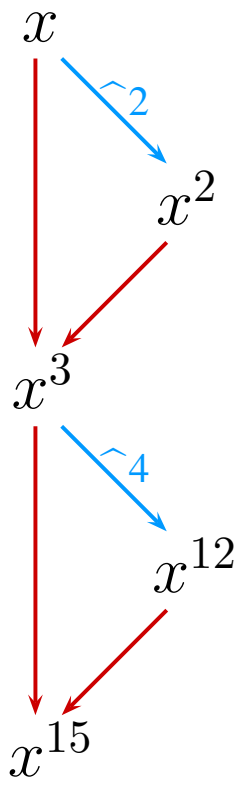
The proposed addition chain:



- one square
- one mult
- one ^4 (two squares)

Masking the S-box

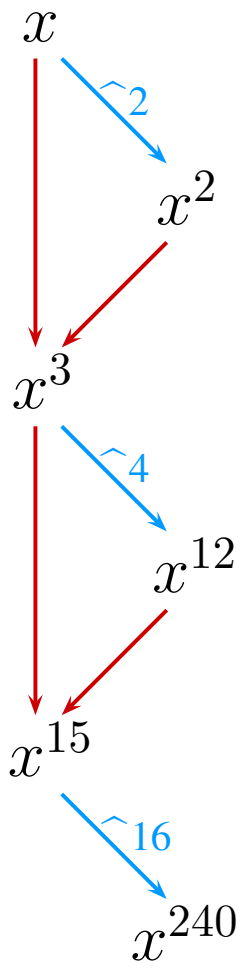
The proposed addition chain:



- one square
- one mult
- one ^4 (two squares)
- one mult

Masking the S-box

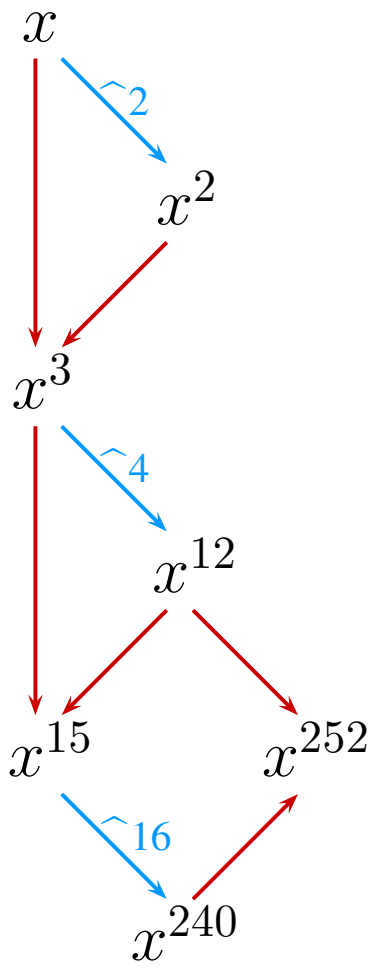
The proposed addition chain:



- one square
- one mult
- one ^4 (two squares)
- one mult
- one ^16 (four squares)

Masking the S-box

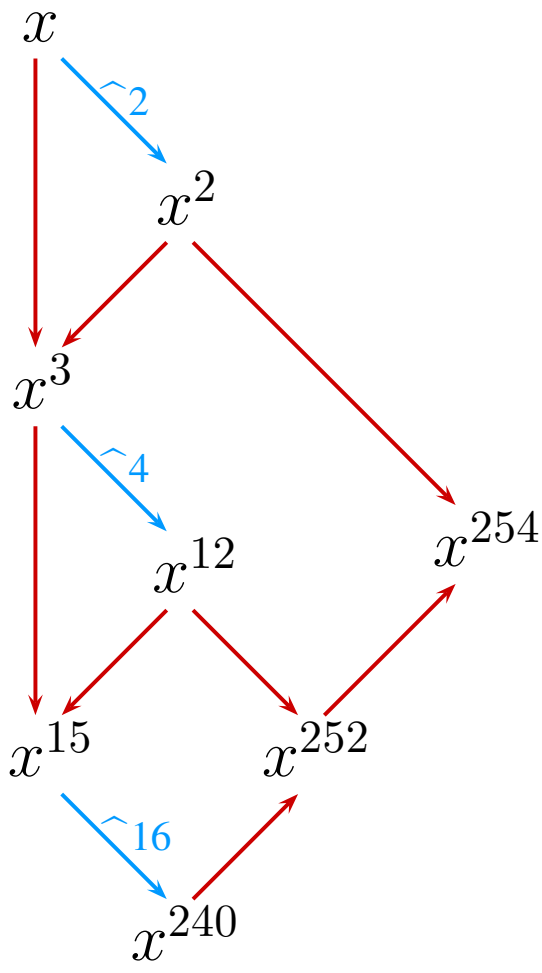
The proposed addition chain:



- one square
- one mult
- one ^ 4 (two squares)
- one mult
- one ^ 16 (four squares)
- one mult

Masking the S-box

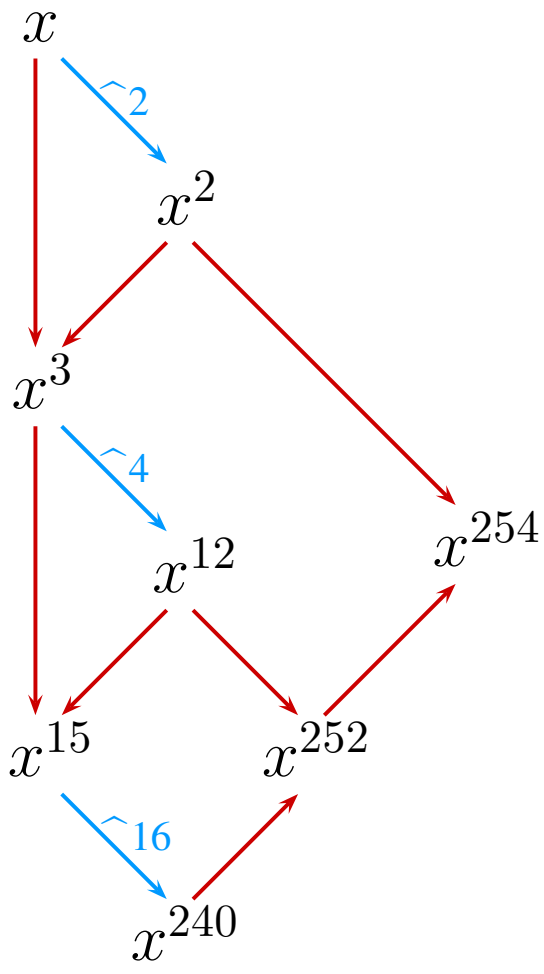
The proposed addition chain:



- one square
- one mult
- one ^ 4 (two squares)
- one mult
- one ^ 16 (four squares)
- one mult
- one mult

Masking the S-box

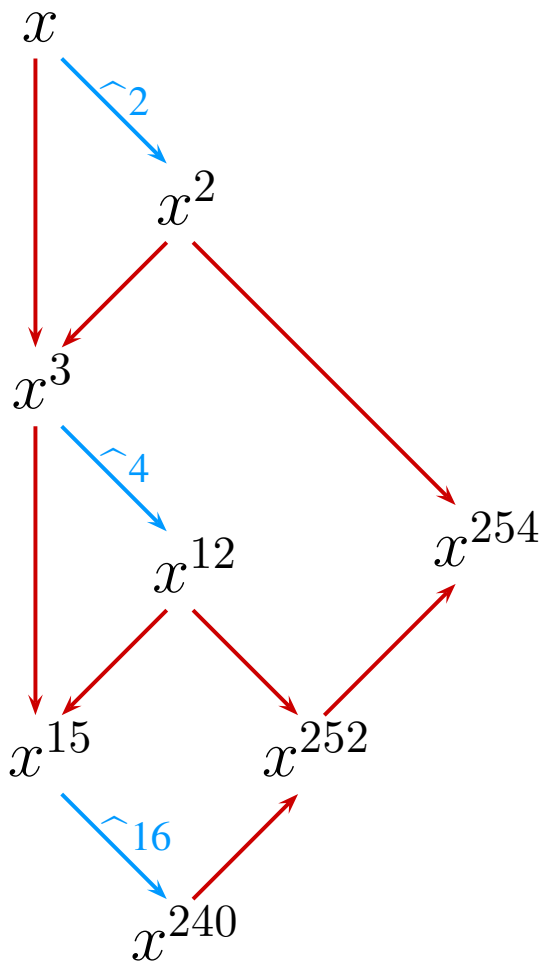
The proposed addition chain:



- one square
- one mult
- one ^ 4 (two squares)
- one mult
- one ^ 16 (four squares)
- one mult
- one mult
- Total: 4 mult and 7 squares

Masking the S-box

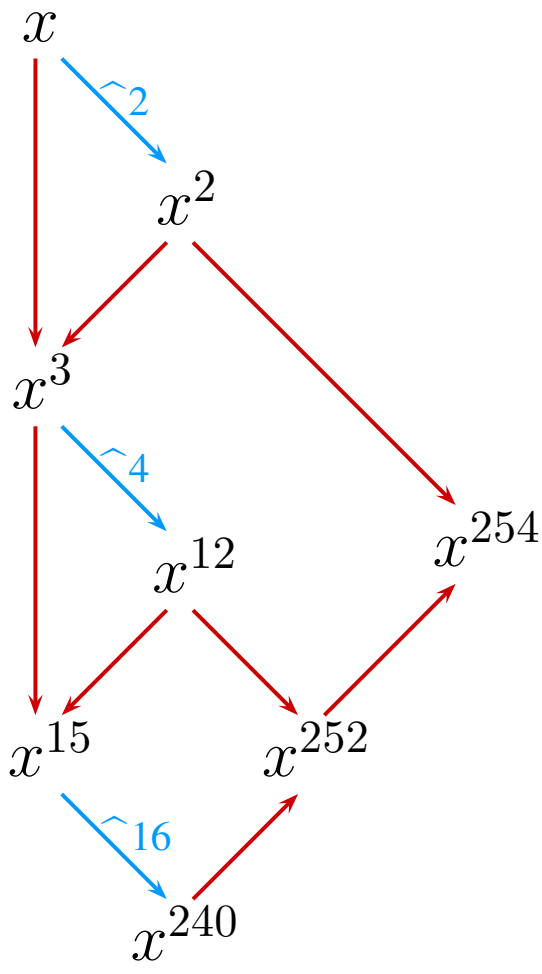
The proposed addition chain:



- one square
- one mult
- one ^4 (two squares)
- one mult
- one ^16 (four squares)
- one mult
- one mult
- Total: 4 mult and 7 squares
- Memory: 3 registers

Masking the S-box

The proposed addition chain:



- one square
- one mult
- one ^4 (two squares)
- one mult
- one ^16 (four squares)
- one mult
- one mult
- Total: 4 mult and 7 squares
- Memory: 3 registers
- LUT for ^2, ^4 and ^16

Masking the S-box

Algorithmic description:

Input: shares x_i s.t. $\bigoplus_i x_i = x$
Output: shares y_i s.t. $\bigoplus_i y_i = x^{254}$

1.	$(z_i)_i \leftarrow (x_i^2)_i$	$[\bigoplus_i z_i = x^2]$
2.	RefreshMasks$\big((z_i)_i\big)$	
3.	$(y_i)_i \leftarrow$ SecMult$\big((z_i)_i, (x_i)_i\big)$	$[\bigoplus_i y_i = x^3]$
4.	$(w_i)_i \leftarrow (y_i^4)_i$	$[\bigoplus_i w_i = x^{12}]$
5.	RefreshMasks$\big((w_i)_i\big)$	
6.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{15}]$
7.	$(y_i)_i \leftarrow (y_i^{16})_i$	$[\bigoplus_i y_i = x^{240}]$
8.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{252}]$
9.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (z_i)_i\big)$	$[\bigoplus_i y_i = x^{254}]$

Masking the S-box

Algorithmic description:

Input: shares x_i s.t. $\bigoplus_i x_i = x$
Output: shares y_i s.t. $\bigoplus_i y_i = x^{254}$

1.	$(z_i)_i \leftarrow (x_i^2)_i$	$[\bigoplus_i z_i = x^2]$
2.	RefreshMasks$\big((z_i)_i\big)$	
3.	$(y_i)_i \leftarrow$ SecMult$\big((z_i)_i, (x_i)_i\big)$	$[\bigoplus_i y_i = x^3]$
4.	$(w_i)_i \leftarrow (y_i^4)_i$	$[\bigoplus_i w_i = x^{12}]$
5.	RefreshMasks$\big((w_i)_i\big)$	
6.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{15}]$
7.	$(y_i)_i \leftarrow (y_i^{16})_i$	$[\bigoplus_i y_i = x^{240}]$
8.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{252}]$
9.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (z_i)_i\big)$	$[\bigoplus_i y_i = x^{254}]$

Masking the S-box

Algorithmic description:

Input: shares x_i s.t. $\bigoplus_i x_i = x$
Output: shares y_i s.t. $\bigoplus_i y_i = x^{254}$

1.	$(z_i)_i \leftarrow (x_i^2)_i$	$[\bigoplus_i z_i = x^2]$
2.	RefreshMasks$\big((z_i)_i\big)$	
3.	$(y_i)_i \leftarrow$ SecMult$\big((z_i)_i, (x_i)_i\big)$	$[\bigoplus_i y_i = x^3]$
4.	$(w_i)_i \leftarrow (y_i^4)_i$	$[\bigoplus_i w_i = x^{12}]$
5.	RefreshMasks$\big((w_i)_i\big)$	
6.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{15}]$
7.	$(y_i)_i \leftarrow (y_i^{16})_i$	$[\bigoplus_i y_i = x^{240}]$
8.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{252}]$
9.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (z_i)_i\big)$	$[\bigoplus_i y_i = x^{254}]$

Masking the S-box

Algorithmic description:

Input: shares x_i s.t. $\bigoplus_i x_i = x$
Output: shares y_i s.t. $\bigoplus_i y_i = x^{254}$

1.	$(z_i)_i \leftarrow (x_i^2)_i$	$[\bigoplus_i z_i = x^2]$
2.	RefreshMasks$\big((z_i)_i\big)$	
3.	$(y_i)_i \leftarrow$ SecMult$\big((z_i)_i, (x_i)_i\big)$	$[\bigoplus_i y_i = x^3]$
4.	$(w_i)_i \leftarrow (y_i^4)_i$	$[\bigoplus_i w_i = x^{12}]$
5.	RefreshMasks$\big((w_i)_i\big)$	
6.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{15}]$
7.	$(y_i)_i \leftarrow (y_i^{16})_i$	$[\bigoplus_i y_i = x^{240}]$
8.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (w_i)_i\big)$	$[\bigoplus_i y_i = x^{252}]$
9.	$(y_i)_i \leftarrow$ SecMult$\big((y_i)_i, (z_i)_i\big)$	$[\bigoplus_i y_i = x^{254}]$

Masking the Whole AES

- Linear operations of encryption/key schedule (ShiftRows, MixColumns, RotWord) processed on every share independently

$$\Lambda\left(\bigoplus_i x_i\right) = \bigoplus_i \Lambda(x_i)$$

Masking the Whole AES

- Linear operations of encryption/key schedule (ShiftRows, MixColumns, RotWord) processed on every share independently

$$\Lambda\left(\bigoplus_i x_i\right) = \bigoplus_i \Lambda(x_i)$$

- Key addition performed by adding each key-share to one single state-share

$$\left(\bigoplus_i s_i\right) \oplus \left(\bigoplus_i k_i\right) = \bigoplus_i (s_i \oplus k_i)$$

Security

dth-order security

$$\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d :$$
$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

Security

dth-order security

$$\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d :$$
$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

- Algorithm split into several transformations applied to one/two dth-order masked value(s)

Security

dth-order security

$$\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d :$$
$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

- Algorithm split into several transformations applied to one/two dth-order masked value(s)
- Every transformation is *locally* secure
  - all transformations are linear (straightforward security) except the field multiplication

Security

dth-order security

$$\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d :$$
$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

- Algorithm split into several transformations applied to one/two dth-order masked value(s)
- Every transformation is *locally* secure
  - all transformations are linear (straightforward security) except the field multiplication
  - field multiplication secured using ISW scheme
  - improved security proof for ISW scheme
    - $d/2 \rightarrow d$

Security

dth-order security

$$\forall (iv_1, iv_2, \ldots, iv_d) \in \{\text{intermediate var. of } \mathcal{E}'\}^d :$$
$$\mathrm{MI}\big((iv_1, iv_2, \ldots, iv_d), (m, k)\big) = 0$$

- Algorithm split into several transformations applied to one/two dth-order masked value(s)
- Every transformation is *locally* secure
  - all transformations are linear (straightforward security) except the field multiplication
  - field multiplication secured using ISW scheme
  - improved security proof for ISW scheme
    - $d/2 \rightarrow d$
- Local security for every transformation implies global security for the whole algorithm

Implementation Results (8051)

Method	K cycles	ms (31MHz)	RAM (bytes)	ROM (bytes)
Unprotected Implementation				
Na.	3	0.1	32	1150
First-Order Masking				
[Messerges FSE'00]	10	0.3	256+35	1553
[Oswald+ FSE'05]	77	2.5	42	3195
Our scheme (d=1)	129	4	73	3153
Second-Order Masking				
[Schramm+ CT-RSA'06]	594	19	512+90	2336
[Rivain+ FSE'08]	672	22	256+86	2215
Our scheme (d=2)	271	9	79	3845
Third-Order Masking				
Our scheme (d=3)	470	15	103	4648

Implementation Results (8051)

Method	K cycles	ms (31MHz)	RAM (bytes)	ROM (bytes)
Unprotected Implementation				
Na.	3	0.1	32	1150
First-Order Masking				
[Messerges FSE'00]	10	0.3	256+35	1553
[Oswald+ FSE'05]	77	2.5	42	3195
Our scheme (d=1)	129	4	73	3153
Second-Order Masking				
[Schramm+ CT-RSA'06]	594	19	512+90	2336
[Rivain+ FSE'08]	672	22	256+86	2215
Our scheme (d=2)	271	9	79	3845
Third-Order Masking				
Our scheme (d=3)	470	15	103	4648

- Interpolation: $30d^2 + 50d + 50$ K cycles
  - $d = 4$: 730 Kc / 24 ms
  - $d = 5$: 1050 Kc / 34 ms

Outline

Conclusion

- First masking scheme for software implementations of AES with provable security at any order
- Based on the work [Ishai-Sahai-Wagner CRYPTO'03]
- Generalization: secure field multiplication in software
- Improved security proof ($d/2 \rightarrow d$), significant in practice
- On-going work:
  - generalization to any S-box/SPN
  - formal security model for dth-order secure implementations