# Algorithmic Approaches to Defeat Side Channel Analysis

Emmanuel PROUFF

ANSSI (French Network and Information Security Agency)

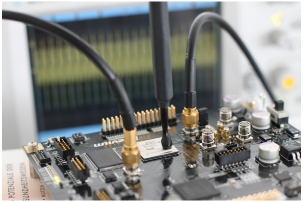
April 13, 2015

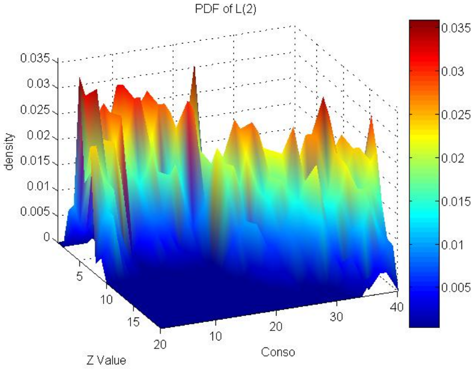
# Probability distribution function (pdf) of Electromagnetic Emanations

$$Z = S(X + k) \text{ with } X = 0 \text{ and } k = 1.$$

# Probability distribution function (pdf) of Electromagnetic Emanations

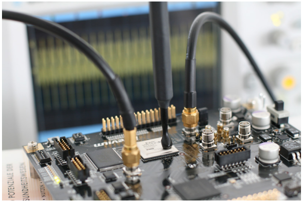
$$Z = S(X + k) \text{ with } X = 0 \text{ and } k = 2.$$

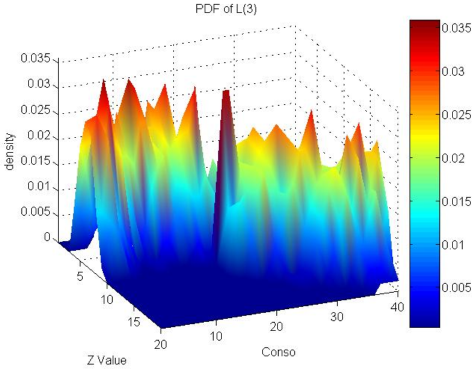




# Probability distribution function (pdf) of Electromagnetic Emanations

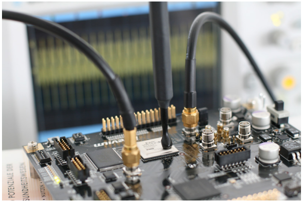
$$Z = S(X + k) \text{ with } X = 0 \text{ and } k = 3.$$

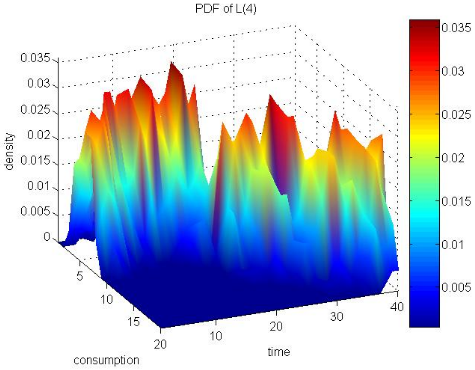




# Probability distribution function (pdf) of Electromagnetic Emanations

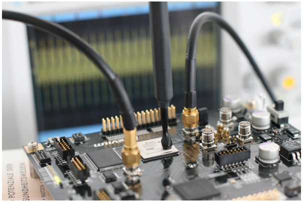
$$Z = S(X + k) \text{ with } X = 0 \text{ and } k = 4.$$

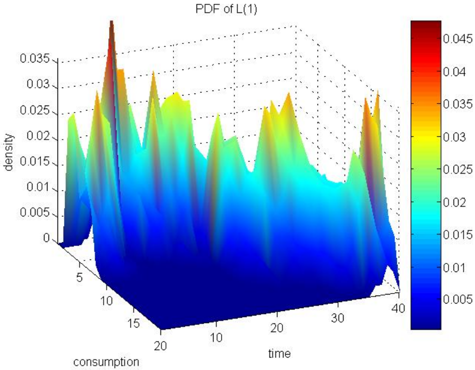




# Probability distribution function (pdf) of Electromagnetic Emanations

$$Z = S(X + k) \text{ with } X = 0 \text{ and } k \in \{1, 2, 3, 4\}.$$





## Side Channel Attacks (SCA)

- Against **each** cryptosystem and **each** implementation, find the most efficient SCA.
  - Efficiency of an SCA?
  - Which attack parameters to improve?
  - SCA common trends?
  - Attacks *versus* Characterization!

## Countermeasures

- For **each** cryptosystem, find efficient/effective countermeasures.
  - Formally define the fact that a countermeasure thwarts an SCA?
  - Which countermeasure for which SCA?
  - What makes a cryptosystem more vulnerable to SCA than another?

- Do we need security proofs?

- Do we need security proofs?
- Yes! Many *ad hoc* security analyses have been invalidated!
  - *e.g.* *GolicTymen02*, *AkkarBevanGoubin2004*, *FumaroliMayerDubois2007*, *CoronProuffRivain2007*, *ProuffMacEvoy2009*, *Debraize2012*, etc.

- Do we need security proofs?
- Yes! Many *ad hoc* security analyses have been invalidated!
  - *e.g.* *GolicTymen02*, *AkkarBevanGoubin2004*, *FumaroliMayerDubois2007*, *CoronProuffRivain2007*, *ProuffMacEvoy2009*, *Debraize2012*, etc.
- Are they sufficient?

- Do we need security proofs?
- Yes! Many *ad hoc* security analyses have been invalidated!
  - *e.g.* *GolicTymen02*, *AkkarBevanGoubin2004*, *FumaroliMayerDubois2007*, *CoronProuffRivain2007*, *ProuffMacEvoy2009*, *Debraize2012*, etc.

- Are they sufficient?
- No! Practical Security $\neq$ Theoretical Security!
  - *e.g.* proofs may be wrong or incomplete
  - or some physical phenomena are difficult to model (*e.g.* glitches)
  - or artefacts in acquisition chain behaviour *MoradiMische2013*

- Do we need security proofs?
- Yes! Many *ad hoc* security analyses have been invalidated!
  - *e.g.* *GolicTymen02*, *AkkarBevanGoubin2004*, *FumaroliMayerDubois2007*, *CoronProuffRivain2007*, *ProuffMacEvoy2009*, *Debraize2012*, etc.
- Are they sufficient?
- No! Practical Security $\neq$ Theoretical Security!
  - *e.g.* proofs may be wrong or incomplete
  - or some physical phenomena are difficult to model (*e.g.* glitches)
  - or artefacts in acquisition chain behaviour *MoradiMische2013*

## An attempt to sum-up

- proofs help **designers** to achieve measurable security
- do not prevent **evaluators** to test theoretically-impossible attacks

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to increase the noise.

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to decrease the SNR.

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to decrease the SNR.
  - increase the noise variance.

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to decrease the SNR.
  - increase the noise variance.
  - force the adversary to himself decrease the SNR.

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to decrease the SNR.
  - increase the noise variance.
  - force the adversary to himself decrease the SNR.

- Secret Sharing: randomly split $Z$ into $d$ shares $Z_1$, ..., $Z_d$:

$$\boxed{Z_1} \quad \boxed{Z_2} \quad \cdots \quad \boxed{Z_d}$$

- Main Remark: SCA efficiency depends on the amount of noise in the observation.

$$L = \varphi(Z) + \underbrace{\mathcal{N}}_{Noise}$$

- Core Idea: define mechanisms to decrease the SNR.
  - increase the noise variance.
  - force the adversary to himself decrease the SNR.

- Secret Sharing: randomly split $Z$ into $d$ shares $Z_1$, ..., $Z_d$:

$$L_1 = \varphi(Z_1) + \mathcal{N}_1 \quad L_2 = \varphi(Z_2) + \mathcal{N}_2 \quad \cdots \quad L_d = \varphi(Z_d) + \mathcal{N}_d$$

  - all the $L_i$ are needed to get information on $Z$!
  - hence the adversary must combine all the $L_i$
  - lead to multiply the $\mathcal{N}_i$ altogether and to merge information and noise in a complex way.

## Adversary Game

In the implementation, find $d$ or less intermediate variables that jointly depend on a secret variable $Z$.

## Developer Game

Translate (Compile?) an implementation into a new one defeating the adversary.

Implementation = sequence of **elementary operations** which **read a memory location** and **write its result in another memory location**.

An approach is to design cryptosystems secure in some leakage models.

An approach is to design cryptosystems secure in some leakage models.

- Recent interest from the crypto theory community (start with *DziembowskiPietrzak2007*).

An approach is to design cryptosystems secure in some leakage models.

- Recent interest from the crypto theory community (start with *DziembowskiPietrzak2007*).
- Proofs are given for some leakage models:
  - Bounded Retrieval Model (BRM): the overall sensitive leakage is bounded.
  - (continuous) Leakage-resilient cryptography (LRC): the leakage is limited for each invocation only.

An approach is to design cryptosystems secure in some leakage models.

- Recent interest from the crypto theory community (start with *DziembowskiPietrzak2007*).
- Proofs are given for some leakage models:
  - Bounded Retrieval Model (BRM): the overall sensitive leakage is bounded.
  - (continuous) Leakage-resilient cryptography (LRC): the leakage is limited for each invocation only.
- BRM primitives are insecure against DPA and its practical relevance is still under discussion.

An approach is to design cryptosystems secure in some leakage models.

- Recent interest from the crypto theory community (start with *DziembowskiPietrzak2007*).
- Proofs are given for some leakage models:
  - Bounded Retrieval Model (BRM): the overall sensitive leakage is bounded.
  - (continuous) Leakage-resilient cryptography (LRC): the leakage is limited for each invocation only.
- BRM primitives are insecure against DPA and its practical relevance is still under discussion.
- LRC primitives aims at DPA-security
  - Based on re-keying techniques
  - The kind of adversary captured by those models is too strong, which strongly impacts the efficiency.

An approach is to design cryptosystems secure in some leakage models.

- Recent interest from the crypto theory community (start with *DziembowskiPietrzak2007*).
- Proofs are given for some leakage models:
  - Bounded Retrieval Model (BRM): the overall sensitive leakage is bounded.
  - (continuous) Leakage-resilient cryptography (LRC): the leakage is limited for each invocation only.
- BRM primitives are insecure against DPA and its practical relevance is still under discussion.
- LRC primitives aims at DPA-security
  - Based on re-keying techniques
  - The kind of adversary captured by those models is too strong, which strongly impacts the efficiency.

Conclusion: need for another approach!

Secure implementations with secret sharing techniques.

Secure implementations with secret sharing techniques.

- First Ideas in *GoubinPatarin99* and *ChariJutlaRaoRohatgi99*.

## Secure implementations with secret sharing techniques.

- First Ideas in *GoubinPatarin99* and *ChariJutlaRaoRohatgi99*.
- Soundness based on the following remark:
  - Bit $x$ masked $\mapsto x_0, x_1, \ldots, x_d$
  - Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$
  - The number of leakage samples to test $\left((L_i)_i | x = 0\right) \stackrel{?}{=} \left((L_i)_i | x = 1\right)$ is lower bounded by $O(1)\sigma^d$.

Secure implementations with secret sharing techniques.

- First Ideas in *GoubinPatarin99* and *ChariJutlaRaoRohatgi99*.
- Soundness based on the following remark:
  - Bit $x$ masked $\mapsto x_0, x_1, \ldots, x_d$
  - Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$
  - The number of leakage samples to test $\big((L_i)_i | x = 0\big) \stackrel{?}{=} \big((L_i)_i | x = 1\big)$ is lower bounded by $O(1)\sigma^d$.
- Until now, two options exist to prove the security:
  - the probing Adversary model
  - the Information Bounded model.

Secure implementations with secret sharing techniques.

- First Ideas in *GoubinPatarin99* and *ChariJutlaRaoRohatgi99*.
- Soundness based on the following remark:
  - Bit $x$ masked $\mapsto x_0, x_1, \ldots, x_d$
  - Leakage : $L_i \sim x_i + \mathcal{N}(\mu, \sigma^2)$
  - The number of leakage samples to test $\big((L_i)_i | x = 0\big) \stackrel{?}{=} \big((L_i)_i | x = 1\big)$ is lower bounded by $O(1)\sigma^d$.
- Until now, two options exist to prove the security:
  - the probing Adversary model
  - the Information Bounded model.
- The two models have been recently unified in *DucDziembowskiFaust14*.

To prove the security of an implementation...

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?
- Ishai-Sahai-Wagner's approach:

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?
- Ishai-Sahai-Wagner's approach:
  - Two players: the **Adversary** who can observe any $d$-tuple of intermediate results and an **Oracle** with no access to the implementation

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?
- Ishai-Sahai-Wagner's approach:
  - Two players: the **Adversary** who can observe any $d$-tuple of intermediate results and an **Oracle** with no access to the implementation
  - The game: prove that, for any $d$-tuple, the oracle can **simulate** the adversary's view of the execution.

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?
- Ishai-Sahai-Wagner's approach:
  - Two players: the **Adversary** who can observe any $d$-tuple of intermediate results and an **Oracle** with no access to the implementation
  - The game: prove that, for any $d$-tuple, the oracle can **simulate** the adversary's view of the execution.
- Method works well for simple schemes (*e.g.* multiplications) BUT difficult to apply in general!

To prove the security of an implementation...

- for $d = 1, 2$: list all the intermediate variables and check that none of them is sensitive.
- for $d \geq 3$: the method above is too costly!
- Issue: how to prove that a scheme can be made $d^{\text{th}}$-order secure for any given $d$?
- Ishai-Sahai-Wagner's approach:
  - Two players: the **Adversary** who can observe any $d$-tuple of intermediate results and an **Oracle** with no access to the implementation
  - The game: prove that, for any $d$-tuple, the oracle can **simulate** the adversary's view of the execution.
- Method works well for simple schemes (*e.g.* multiplications) BUT difficult to apply in general!
- Recently *Belaid, Fouque and Barthe* developed automatic tools to generate security certificates.

- Implementation Model. *Micali-Reyzin, TCC 2004*

Implementation = seq. of elem. computations producing a list of interm. results $(Z_i)_i$.

- Implementation Model. *Micali-Reyzin, TCC 2004*

Implementation $=$ seq. of elem. computations producing a list of interm. results $(Z_i)_i$.

- Leakage on $Z_i$ modelled by a probabilistic function $f_i$ s.t.

$$\mathrm{MI}(Z_i; f_i(Z_i)) \leq O(1/\psi) \ ,$$

where $\psi$ is a security parameter depending on the noise.

- Implementation Model. *Micali-Reyzin, TCC 2004*

$$\text{Implementation} = \begin{array}{l}\text{seq. of elem. computations producing}\\ \text{a list of interm. results } (Z_i)_i.\end{array}$$

- Leakage on $Z_i$ modelled by a probabilistic function $f_i$ s.t.

$$\mathrm{MI}(Z_i; f_i(Z_i)) \leq O(1/\psi) \;,$$

where $\psi$ is a security parameter depending on the noise.

- Security Proof goal: find a deterministic function $P$ s.t.:

$$\mathrm{MI}((X,k); (f_i(Z_i))_i) \leq P(1/\psi)$$

where $X$ is the plaintext and $k$ is the key.

- First Issue: how to share sensitive data?

- Second Issue: how to securely process on shared data?

- First Issue: how to share sensitive data?
- Related to:
  - secret sharing *Shamir79*
  - design of error correcting codes with large dual distance *Massey93*

- Second Issue: how to securely process on shared data?
- Related to:
  - secure multi-party computation *NikovaRijmenSchläffer2008 ProuffRoche2011*
  - circuit processing in presence of leakage *GoldwasserRothblum2012*
  - efficient polynomial evaluation *CarletGoubinProuffQuisquater-Rivain2012,CoronProuffRoche2012*

- Linear Secret Sharing with parameters $n$ and $d$:
  - $n$ elements $Z_i$ such that

$$Z = \sum_i Z_i$$

  - no sub-family of $d-1$ $Z_i$ depends on $Z$.

- Linear Secret Sharing with parameters $n$ and $d$:
  - $n$ elements $Z_i$ such that

$$Z = \sum_i Z_i$$

  - no sub-family of $d-1$ $Z_i$ depends on $Z$.

- Massey (1993):

designing an $(n, d)$ linear secret sharing

$$\Longleftrightarrow$$

building a code with length $n+1$ and dual distance $d$

- Linear Secret Sharing with parameters $n$ and $d$:
  - $n$ elements $Z_i$ such that

$$Z = \sum_i Z_i$$

  - no sub-family of $d-1$ $Z_i$ depends on $Z$.

- Massey (1993):

designing an $(n,d)$ linear secret sharing

$$\Longleftrightarrow$$

building a code with length $n+1$ and dual distance $d$

- Yes, interesting, but ... who cares?

- Linear Secret Sharing with parameters $n$ and $d$:
  - $n$ elements $Z_i$ such that

$$Z = \sum_i Z_i$$

  - no sub-family of $d-1$ $Z_i$ depends on $Z$.

- Massey (1993):

designing an $(n, d)$ linear secret sharing

$$\Longleftrightarrow$$

building a code with length $n+1$ and dual distance $d$

- Yes, interesting, but ... who cares?
  - gives a general framework to describe and analyse all linear sharing schemes

- Linear Secret Sharing with parameters $n$ and $d$:
  - $n$ elements $Z_i$ such that

$$Z = \sum_i Z_i$$

  - no sub-family of $d-1$ $Z_i$ depends on $Z$.

- Massey (1993):

designing an $(n,d)$ linear secret sharing

$$\Longleftrightarrow$$

building a code with length $n+1$ and dual distance $d$

- Yes, interesting, but ... who cares?
  - gives a general framework to describe and analyse all linear sharing schemes
  - links our problems with those of a rich community

- Linear Sharing = Encoding

$$\begin{pmatrix} Z & R_1 & \dots & R_{k-1} \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 & \alpha_{1,k} & \dots & \alpha_{1,n} \\ 0 & 1 & 0 & 0 & \alpha_{2,k} & \dots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & \alpha_{k,k} & \dots & \alpha_{k,n} \end{pmatrix} = \begin{pmatrix} Z & Z_1 & \dots & Z_{k-1} & Z_k & \dots & Z_n \end{pmatrix}$$

- Linear Sharing = Encoding

$$
\begin{pmatrix} {\color{red}Z} & Z_1 & \dots & Z_n \end{pmatrix} \times \begin{pmatrix} \alpha_{1,k} & \dots & \dots & \alpha_{k,k} \\ \alpha_{1,k+1} & \dots & \dots & \alpha_{k,k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{1,n} & \dots & \dots & \alpha_{k,n} \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -1 \end{pmatrix}
$$

$$
= \begin{pmatrix} 0 & \dots & 0 \end{pmatrix}
$$

- Linear Sharing = Encoding

$$\begin{pmatrix} Z & Z_1 & \dots & Z_n \end{pmatrix} \times \begin{pmatrix} \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 \end{pmatrix}$$

- Linear Sharing = Encoding

$$\begin{pmatrix} Z & Z_1 & \dots & Z_n \end{pmatrix} \times \begin{pmatrix} \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 \end{pmatrix}$$

- implies for every $i \in [1..k]$:

$$Z = H_{i0}^{-1} \sum_{j=2}^{n} Z_j \times H_{i,j} \quad .$$

- Linear Sharing = Encoding

$$\begin{pmatrix} Z & Z_1 & \dots & Z_n \end{pmatrix} \times \begin{pmatrix} \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 \end{pmatrix}$$

- implies for every $i \in [1..k]$:

$$Z = H_{i0}^{-1} \sum_{j=2}^{n} Z_j \times H_{i,j} \quad .$$

- masking order $< \min_i \mathrm{HW}(\vec{H}_i) - 1$

- Linear Sharing = Encoding

$$\begin{pmatrix} Z & Z_1 & \dots & Z_n \end{pmatrix} \times \begin{pmatrix} \vec{H}_1 & \vec{H}_2 & \dots & \vec{H}_k \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 \end{pmatrix}$$

- implies for every $i \in [1..k]$:

$$Z = H_{i0}^{-1} \sum_{j=2}^{n} Z_j \times H_{i,j} \ .$$

- masking order $< \min_i \mathrm{HW}(\vec{H}_i) - 1$
- Actually masking order $= \min_{\vec{H} \in C^{\perp}} \mathrm{HW}(\vec{H}) - 1$ *Massey93*

- Boolean Sharing: encoding with the matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

implies $k = n - 1$.

- Boolean Sharing: encoding with the matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

  implies $k = n - 1$.
- Shamir's secret Sharing:
  - generate a random degree-$d$ polynomial $P(X)$ such that $P(0) = Z$
  - build the $Z_i$ such that $Z_i = P(\alpha_i)$ for $n \geq 2d$ different public values $\alpha_i$.

- Boolean Sharing: encoding with the matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

  implies $k = n - 1$.
- Shamir's secret Sharing:
  - generate a random degree-$d$ polynomial $P(X)$ such that $P(0) = Z$
  - build the $Z_i$ such that $Z_i = P(\alpha_i)$ for $n \geq 2d$ different public values $\alpha_i$.
- ... amounts to define a Reed-Solomon code with parameters $[n+1, d+1, \cdot]$ *McElieceSarwate81*.

- Boolean Sharing: encoding with the matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

  implies $k = n - 1$.
- Shamir's secret Sharing:
  - generate a random degree-$d$ polynomial $P(X)$ such that $P(0) = Z$
  - build the $Z_i$ such that $Z_i = P(\alpha_i)$ for $n \geq 2d$ different public values $\alpha_i$.
- ... amounts to define a Reed-Solomon code with parameters $[n+1, d+1, \cdot]$ *McElieceSarwate81*.
- Main issue: minimize $n$ for a given $d$.

- Multiplicative Masking *Gollic2002, GenelleProuffQuisquater2010*

$$Z \mapsto Z_0, \ldots, Z_d \text{ s.t. } Z_i \neq 0 \text{ and } Z = Z_0 \times \cdots \times Z_d$$

- Multiplicative Masking *Gollic2002, GenelleProuffQuisquater2010*

$$Z \mapsto Z_0, \ldots, Z_d \text{ s.t. } Z_i \neq 0 \text{ and } Z = Z_0 \times \cdots \times Z_d$$

- Affine Masking *vonWillich2001,FumarolliMartinelliProuffRivain2010*

$$Z \mapsto Z_0, Z_1, Z_2 \text{ s.t. } Z_1 \neq 0 \text{ and } Z = \frac{Z_0}{Z_1} + Z_2$$

- Multiplicative Masking *Gollic2002, GenelleProuffQuisquater2010*

$$Z \mapsto Z_0, \ldots, Z_d \text{ s.t. } Z_i \neq 0 \text{ and } Z = Z_0 \times \cdots \times Z_d$$

- Affine Masking *vonWillich2001,FumarolliMartinelliProuffRivain2010*

$$Z \mapsto Z_0, Z_1, Z_2 \text{ s.t. } Z_1 \neq 0 \text{ and } Z = \frac{Z_0}{Z_1} + Z_2$$

- Modular Additive Masking *Coron1999*

$$Z \mapsto Z_0, Z_1 \text{ s.t. } Z = Z_1 + Z_2 \mod \ldots$$

- Multiplicative Masking *Gollic2002, GenelleProuffQuisquater2010*

$$Z \mapsto Z_0, \dots, Z_d \text{ s.t. } Z_i \neq 0 \text{ and } Z = Z_0 \times \cdots \times Z_d$$

- Affine Masking *vonWillich2001,FumarolliMartinelliProuffRivain2010*

$$Z \mapsto Z_0, Z_1, Z_2 \text{ s.t. } Z_1 \neq 0 \text{ and } Z = \frac{Z_0}{Z_1} + Z_2$$

- Modular Additive Masking *Coron1999*

$$Z \mapsto Z_0, Z_1 \text{ s.t. } Z = Z_1 + Z_2 \mod \dots$$

- Homographic Masking *CourtoisGoubin2005*

$$Z \mapsto \frac{Z_0 \times Z + Z_1}{Z_2 \times Z + Z_3} \text{ or } \infty \text{ if } Z = -\frac{Z_3}{Z_2} \text{ or } \frac{Z_0}{Z_2} \text{ if } Z = \infty$$

- Leakage Squeezing

*MaghrebiGuilleyDanger2011,CarletDangerGuilleyMaghrebi2014*

$$Z \mapsto Z_0, Z_1 \text{ s.t. } Z = Z_0 \oplus Z_1 \text{ and } Z_i \in \mathcal{C}$$

where $\mathcal{C}$ is a code with dual distance $d$.

- Leakage Squeezing

  *MaghrebiGuilleyDanger2011,CarletDangerGuilleyMaghrebi2014*

$$Z \mapsto Z_0, Z_1 \text{ s.t. } Z = Z_0 \oplus Z_1 \text{ and } Z_i \in \mathcal{C}$$

  where $\mathcal{C}$ is a code with dual distance $d$.

- Inner Product *BalaschFaustGierlichsVerbauwhede2012* and *BalaschFaustGierlichs2015*

$$Z \mapsto \mathbf{L}, \mathbf{R} \in \mathrm{GF}(2^n)^d \text{ s.t. } Z = \mathbf{L} \cdot \mathbf{R}$$

- Securing elementary Operations:

- Securing elementary Operations:
- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)

- Securing elementary Operations:
- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)
- Extended to any field in *RivainProuff2010* and *FaustRabinReyzinTromerVaikuntanathan2011*

- Securing elementary Operations:
- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)
- Extended to any field in *RivainProuff2010* and *FaustRabinReyzinTromerVaikuntanathan2011*
- Based on Boolean Sharing: $Z = Z_0 \oplus Z_1 \oplus \ldots Z_d$

- Securing elementary Operations:
- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)
- Extended to any field in *RivainProuff2010* and *FaustRabinReyzinTromerVaikuntanathan2011*
- Based on Boolean Sharing: $Z = Z_0 \oplus Z_1 \oplus \ldots Z_d$
- Securing linear functions L:

$$\begin{array}{cccc} Z_0 & Z_1 & \cdots & Z_d \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \mathrm{L}(Z_0) & \mathrm{L}(Z_1) & \cdots & \mathrm{L}(Z_d) \end{array}$$

- Securing elementary Operations:
- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)
- Extended to any field in *RivainProuff2010* and *FaustRabinReyzinTromerVaikuntanathan2011*
- Based on Boolean Sharing: $Z = Z_0 \oplus Z_1 \oplus \dots Z_d$
- Securing linear functions L:

$$\begin{array}{ccccccccc} & Z_0 & & Z_1 & & \cdots & & Z_d \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \mathrm{L}(Z) = & \mathrm{L}(Z_0) & \oplus & \mathrm{L}(Z_1) & \oplus & \cdots & \oplus & \mathrm{L}(Z_d) \end{array}$$

- Securing elementary Operations:

- Original idea by *Ishai-Sahai-Wagner*: limited to GF(2)
- Extended to any field in *RivainProuff2010* and *FaustRabinReyzinTromerVaikuntanathan2011*
- Based on Boolean Sharing: $Z = Z_0 \oplus Z_1 \oplus \ldots Z_d$
- Securing linear functions L:

$$\begin{array}{cccccccc} & Z_0 & & Z_1 & & \cdots & & Z_d \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \mathrm{L}(Z) = & \mathrm{L}(Z_0) & \oplus & \mathrm{L}(Z_1) & \oplus & \cdots & \oplus & \mathrm{L}(Z_d) \end{array}$$

- Much more difficult for non-linear functions (*i.e.* multiplication)

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

## Securing Multiplication *IshaiSahaiWagner2003*:

- 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. $\bigoplus_i c_i = ab$

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

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

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

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

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

where the $r_{i,j}$ are a $((d+1)^2, d)$-sharing of $0$.

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

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

- Securing Multiplication *IshaiSahaiWagner2003*:

  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

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

$c_1$

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

$$\begin{array}{ccc} \multicolumn{3}{c}{\begin{pmatrix} a_0b_0 \oplus r_{0,0} & a_0b_1 \oplus r_{0,1} & a_0b_2 \oplus r_{0,2} \\ a_1b_0 \oplus r_{1,0} & a_1b_1 \oplus r_{1,1} & a_1b_2 \oplus r_{1,2} \\ a_2b_0 \oplus r_{2,0} & a_2b_1 \oplus r_{2,1} & a_2b_2 \oplus r_{2,2} \end{pmatrix}} \\ c_1 & c_2 & \end{array}$$

- Securing Multiplication *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- Illustration of ISW scheme for $d = 2$:

$$\begin{array}{ccc} \multicolumn{3}{c}{\begin{pmatrix} a_0b_0 \oplus r_{0,0} & a_0b_1 \oplus r_{0,1} & a_0b_2 \oplus r_{0,2} \\ a_1b_0 \oplus r_{1,0} & a_1b_1 \oplus r_{1,1} & a_1b_2 \oplus r_{1,2} \\ a_2b_0 \oplus r_{2,0} & a_2b_1 \oplus r_{2,1} & a_2b_2 \oplus r_{2,2} \end{pmatrix}} \\ c_1 & c_2 & c_3 \end{array}$$

- **Securing Multiplication** *IshaiSahaiWagner2003*:
  - 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. $\bigoplus_i c_i = ab$

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

- **Illustration of ISW scheme for $d = 2$:**

$$\begin{array}{ccc} \multicolumn{3}{c}{\begin{pmatrix} a_0b_0 \oplus r_{0,0} & a_0b_1 \oplus r_{0,1} & a_0b_2 \oplus r_{0,2} \\ a_1b_0 \oplus r_{1,0} & a_1b_1 \oplus r_{1,1} & a_1b_2 \oplus r_{1,2} \\ a_2b_0 \oplus r_{2,0} & a_2b_1 \oplus r_{2,1} & a_2b_2 \oplus r_{2,2} \end{pmatrix}} \\ c_1 & c_2 & c_3 \end{array}$$

- Actually, we can do it with $(d+1)^2/2$ random values instead of $(d+1)^2$.

## Securing any Polynomial evaluation

- Write the s-box $\mathrm{S} : \{0,1\}^n \to \{0,1\}^m$ as a polynomial function over $\mathrm{GF}(2^n)$:

$$\mathrm{S}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{2^n-1} x^{2^n-1}$$

# Securing any Polynomial evaluation

- Write the s-box $\mathrm{S} : \{0,1\}^n \to \{0,1\}^m$ as a polynomial function over $\mathrm{GF}(2^n)$:

$$\mathrm{S}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{2^n-1} x^{2^n-1}$$

- Four kinds of operations over $\mathrm{GF}(2^n)$:
  1. additions
  2. scalar multiplications (*i.e.* by constants)
  3. squares
  4. regular multiplications

# Securing any Polynomial evaluation

- Write the s-box $\mathrm{S} : \{0,1\}^n \to \{0,1\}^m$ as a polynomial function over $\mathrm{GF}(2^n)$:

$$\mathrm{S}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{2^n-1} x^{2^n-1}$$

- Four kinds of operations over $\mathrm{GF}(2^n)$:
  1. additions
  2. scalar multiplications (*i.e.* by constants)
  3. squares
  4. regular multiplications

- Schemes with complexity $O(d)$ for the 3 first kinds
  - $(x+y) \longrightarrow (x_0+y_0), (x_1+y_1), \cdots, (x_d+y_d)$
  - $x^2 \longrightarrow x_0^2, x_1^2, \cdots + x_d^2$
  - $a \cdot x \longrightarrow a \cdot x_0, a \cdot x_1, \cdots, a \cdot x_d$

## Securing any Polynomial evaluation

- Write the s-box $\mathrm{S} : \{0,1\}^n \to \{0,1\}^m$ as a polynomial function over $\mathrm{GF}(2^n)$:

$$\mathrm{S}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{2^n-1} x^{2^n-1}$$

- Four kinds of operations over $\mathrm{GF}(2^n)$:
  1. additions
  2. scalar multiplications (*i.e.* by constants)
  3. squares
  4. regular multiplications $\Rightarrow$ *nonlinear multiplications*

- Schemes with complexity $O(d)$ for the 3 first kinds
  - $(x+y) \longrightarrow (x_0+y_0), (x_1+y_1), \cdots, (x_d+y_d)$
  - $x^2 \longrightarrow x_0^2, x_1^2, \cdots + x_d^2$
  - $a \cdot x \longrightarrow a \cdot x_0, a \cdot x_1, \cdots, a \cdot x_d$

- Schemes with complexity $O(d^2)$ for the non-linear multiplication

  *IshaiSahaiWagner2004*

**Definition** (*CarletGoubinProuffQuisquaterRivain2012*)

The masking complexity of $S$ is the minimal number of non-linear multiplications needed for its evaluation.

**Definition** (*CarletGoubinProuffQuisquaterRivain2012*)

The masking complexity of $S$ is the minimal number of non-linear multiplications needed for its evaluation.

Problematic 1: compute the masking complexity of any $S$ (at least bounds).

**Definition** (*CarletGoubinProuffQuisquaterRivain2012*)

The masking complexity of $S$ is the minimal number of non-linear multiplications needed for its evaluation.

Problematic 1: compute the masking complexity of any $S$ (at least bounds).

Problematic 2: find evaluations methods efficient for the masking complexity criterion.

**Definition** (*CarletGoubinProuffQuisquaterRivain2012*)

The masking complexity of $S$ is the minimal number of non-linear multiplications needed for its evaluation.

Problematic 1: compute the masking complexity of any $S$ (at least bounds).

Problematic 2: find evaluations methods efficient for the masking complexity criterion.

For monomials: amounts to look for short 2-addition-chain exponentiations.

**Definition** (*CarletGoubinProuffQuisquaterRivain2012*)

The masking complexity of $S$ is the minimal number of non-linear multiplications needed for its evaluation.

Problematic 1: compute the masking complexity of any $S$ (at least bounds).

Problematic 2: find evaluations methods efficient for the masking complexity criterion.

For monomials: amounts to look for short 2-addition-chain exponentiations.

For polynomials: amounts to find efficient decompositions;

- Knuth-Eve algorithm *VonZurGathenNoker2003*
- or the Cyclotomic Method *CarletGoubinProuffQuisquaterRivain2012*
- or Coron-Roy-Vivek's method *CoronRoyVivek2014*

- Idea: Mix additive with multiplicative masking defined on the same field.

- Idea: Mix additive with multiplicative masking defined on the same field.
- Recall (Additive masking):
  $x \in \mathrm{GF}(2^n) \mapsto (x_0, \cdots x_d) \in \mathrm{GF}(2^n)^{d+1}$ s.t.

$$\sum_i x_i = x \quad .$$

- Idea: Mix additive with multiplicative masking defined on the same field.
- Recall (Additive masking):
  $x \in \mathrm{GF}(2^n) \mapsto (x_0, \cdots x_d) \in \mathrm{GF}(2^n)^{d+1}$ s.t.
  $$\sum_i x_i = x \ .$$
- Recall (Multiplicative masking):
  $x \in \mathrm{GF}(2^n)^* \mapsto (x_0, \cdots x_d) \in \mathrm{GF}(2^n)^{*d+1}$ s.t.
  $$\prod_i x_i = x \ .$$

- Idea: Mix additive with multiplicative masking defined on the same field.
- Recall (Additive masking):
$x \in \mathrm{GF}(2^n) \mapsto (x_0, \cdots x_d) \in \mathrm{GF}(2^n)^{d+1}$ s.t.
$$\sum_i x_i = x \ .$$
- Recall (Multiplicative masking):
$x \in \mathrm{GF}(2^n)^* \mapsto (x_0, \cdots x_d) \in \mathrm{GF}(2^n)^{*d+1}$ s.t.
$$\prod_i x_i = x \ .$$
- So, use additive masking for affine transformations and multiplicative masking for power functions.

- Issue 1: convert additive masking into multiplicative masking without leaking information in the $d^{\text{th}}$-order probing model?

- Issue 1: convert additive masking into multiplicative masking without leaking information in the $d^{\text{th}}$-order probing model?
  - Solution: conversions algorithms proposed in *GenelleProuffQuisquater11* (complexity: $d^2$ additions and $d(3+d)/2$ multiplications).

- Issue 1: convert additive masking into multiplicative masking without leaking information in the $d^{\text{th}}$-order probing model?
  - Solution: conversions algorithms proposed in *GenelleProuffQuisquater11* (complexity: $d^2$ additions and $d(3+d)/2$ multiplications).

- Issue 2: multiplicative is only sound in the multiplicative group! How to deal with the 0 value problem?

- Issue 1: convert additive masking into multiplicative masking without leaking information in the $d^{\text{th}}$-order probing model?
  - Solution: conversions algorithms proposed in *GenelleProuffQuisquater11* (complexity: $d^2$ additions and $d(3+d)/2$ multiplications).

- Issue 2: multiplicative is only sound in the multiplicative group! How to deal with the 0 value problem?
  - Solution: map the sharing of 0 into the sharing of 1 and keep trace of this modification for further correction.
  - Amounts to secure the processing of the function

$$x \mapsto x \oplus \delta_0(x) \text{ with } \delta_0(x) = x_0 \texttt{ AND } x_1 \texttt{ AND } \ldots \texttt{ AND } x_n \ .$$

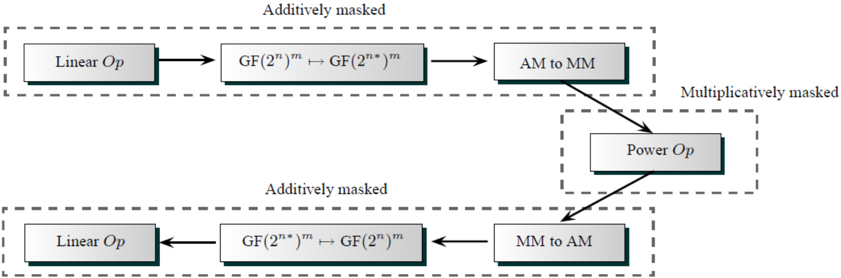
- Issue 1: convert additive masking into multiplicative masking without leaking information in the $d^{\text{th}}$-order probing model?
  - Solution: conversions algorithms proposed in *GenelleProuffQuisquater11* (complexity: $d^2$ additions and $d(3+d)/2$ multiplications).

- Issue 2: multiplicative is only sound in the multiplicative group! How to deal with the 0 value problem?
  - Solution: map the sharing of 0 into the sharing of 1 and keep trace of this modification for further correction.
  - Amounts to secure the processing of the function

$$x \mapsto x \oplus \delta_0(x) \text{ with } \delta_0(x) = x_0 \texttt{ AND } x_1 \texttt{ AND } \dots \texttt{ AND } x_n \ .$$

  - Soundness: for any power $e$, we have

$$(x \oplus \delta_0(x))^e = x^e \oplus \delta_0(x)$$


Additively masked
Linear $Op$
$GF(2^n)^m \mapsto GF(2^{n*})^m$
AM to MM
Multiplicatively masked
Power $Op$
Additively masked
Linear $Op$
$GF(2^{n*})^m \mapsto GF(2^n)^m$
MM to AM

from *NikovaRijmenSchlaffer2008*

from *NikovaRijmenSchlaffer2008*

## Notation:

$$S^\star(Z_0, \cdots, Z_d) \doteq S(\sum_i Z_i) = S(Z)$$

from *NikovaRijmenSchlaffer2008*

Notation:

$$S^{\star}(Z_0, \cdots, Z_d) \doteq S(\sum_i Z_i) = S(Z)$$

Idea: find the smallest $t$ s.t. there exist $t$ indices subsets $I_j \subsetneq \{0, ..., d+1\}$ and $t$ balanced functions $S_j$ s.t.:

from *NikovaRijmenSchlaffer2008*

Notation:

$$S^{\star}(Z_0, \cdots, Z_d) \doteq S(\sum_i Z_i) = S(Z)$$

Idea: find the smallest $t$ s.t. there exist $t$ indices subsets $I_j \subsetneq \{0, ..., d+1\}$ and $t$ balanced functions $S_j$ s.t.:

1. **[Completeness]**

$$S(Z_0, \cdots, Z_d) = \sum_j S_j((Z_i)_{i \in I_j}) \ ,$$

from NikovaRijmenSchlaffer2008

Notation:

$$S^\star(Z_0,\cdots,Z_d) \doteq S(\sum_i Z_i) = S(Z)$$

Idea: find the smallest $t$ s.t. there exist $t$ indices subsets $I_j \subsetneq \{0,...,d+1\}$ and $t$ balanced functions $S_j$ s.t.:

1. **[Completeness]**

$$S(Z_0,\cdots,Z_d) = \sum_j S_j((Z_i)_{i\in I_j}) \ ,$$

2. **[Security]** The $t$ values $S_0((Z_i)_{i\in I_0})$, ..., $S_{t-1}((Z_i)_{i\in I_{t-1}})$ form a $t$-sharing of $S(Z)$.

from *NikovaRijmenSchlaffer2008*

Notation:

$$S^\star(Z_0,\cdots,Z_d) \doteq S(\sum_i Z_i) = S(Z)$$

Idea: find the smallest $t$ s.t. there exist $t$ indices subsets $I_j \subsetneq \{0,\ldots,d+1\}$ and $t$ balanced functions $S_j$ s.t.:

1. **[Completeness]**

$$S(Z_0,\cdots,Z_d) = \sum_j S_j((Z_i)_{i\in I_j}) \ ,$$

2. **[Security]** The $t$ values $S_0((Z_i)_{i\in I_0})$, ..., $S_{t-1}((Z_i)_{i\in I_{t-1}})$ form a $t$-sharing of $S(Z)$.

Recently extended to any order *at Asiacrypt2014*.

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

## Secure Evaluation of a Polynomial $h(x)$ with algebraic degree $s$

$h(x)$ a polynomial with algebraic degree $s$

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{j \le s} c_j \sum_{\substack{I \subseteq [1;d] \\ |I| = j}} h\Big(\sum_{i \in I} a_i\Big) \ ,$$

where $c_j$ are constant binary coefficients.

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

**Secure Evaluation of a Polynomial $h(x)$ with algebraic degree $s$**

$h(x)$ a polynomial with algebraic degree $s$

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{j \leq s} c_j \sum_{\substack{I \subseteq [1;d] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \; ,$$

where $c_j$ are constant binary coefficients.

Hence: securing at order $d$ reduces to securing at order $s$.

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

## Secure Evaluation of a Polynomial $h(x)$ with algebraic degree $s$

$h(x)$ a polynomial with algebraic degree $s$

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{j \le s} c_j \sum_{\substack{I \subseteq [1;d] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \ ,$$

where $c_j$ are constant binary coefficients.

Hence: securing at order $d$ reduces to securing at order $s$.
Leads to the secure evaluation methods with complexity $O(d^s)$.

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

## Secure Evaluation of a Polynomial $h(x)$ with algebraic degree $s$

$h(x)$ a polynomial with algebraic degree $s$

$$h\left(\sum_{i=1}^{d} a_i\right) = \sum_{j\leq s} c_j \sum_{\substack{I\subseteq[1;d]\\ |I|=j}} h\left(\sum_{i\in I} a_i\right) ,$$

where $c_j$ are constant binary coefficients.

Hence: securing at order $d$ reduces to securing at order $s$.
Leads to the secure evaluation methods with complexity $O(d^s)$.
Example: securing degree-2 functions is as complex as securing a multiplication (with ISW scheme).

algebraic degree of a polynomial: greatest Hamming weight of the power of its monomials (with non-zero coefficients).

## Secure Evaluation of a Polynomial $h(x)$ with algebraic degree $s$

$h(x)$ a polynomial with algebraic degree $s$

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{j \leq s} c_j \sum_{\substack{I \subseteq [1;d] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \ ,$$

where $c_j$ are constant binary coefficients.

Hence: securing at order $d$ reduces to securing at order $s$.
Leads to the secure evaluation methods with complexity $O(d^s)$.
Example: securing degree-2 functions is as complex as securing a multiplication (with ISW scheme).
Efficient (compared to SoA) for small $s$ or $n \ll d^s$.

Extend CRV's method and exchange nonlinear multiplications for evaluations of degree-$s$ functions (with $s$ small).

Extend CRV's method and exchange nonlinear multiplications for evaluations of degree-$s$ functions (with $s$ small).

1. Randomly generate $r$ degree-$s$ polynomials $f_i$

Extend CRV's method and exchange nonlinear multiplications for evaluations of degree-$s$ functions (with $s$ small).

1. Randomly generate $r$ degree-$s$ polynomials $f_i$
2. Derive new polynomials $(g_i)_i$:

$$\begin{cases} g_1(x) = f_1(x) \\ g_i(x) = f_i\big(g_{i-1}(x)\big) \end{cases}$$

Extend CRV's method and exchange nonlinear multiplications for evaluations of degree-$s$ functions (with $s$ small).

1. Randomly generate $r$ degree-$s$ polynomials $f_i$
2. Derive new polynomials $(g_i)_i$:

$$\begin{cases} g_1(x) = f_1(x) \\ g_i(x) = f_i\big(g_{i-1}(x)\big) \end{cases}$$

3. Randomly generate $t$ polynomials $(q_i)_i$ s.t.

$$q_i(x) = \sum_{j=1}^{r} \ell_{i,j}\big(g_j(x)\big) + \ell_{i,0}(x) \ ,$$

where the $\ell_j$ are linearized polynomials.

Extend CRV's method and exchange nonlinear multiplications for evaluations of degree-$s$ functions (with $s$ small).

1. Randomly generate $r$ degree-$s$ polynomials $f_i$
2. Derive new polynomials $(g_i)_i$:

$$\begin{cases} g_1(x) = f_1(x) \\ g_i(x) = f_i\big(g_{i-1}(x)\big) \end{cases}$$

3. Randomly generate $t$ polynomials $(q_i)_i$ s.t.

$$q_i(x) = \sum_{j=1}^{r} \ell_{i,j}\big(g_j(x)\big) + \ell_{i,0}(x) \ ,$$

   where the $\ell_j$ are linearized polynomials.
4. Find $t$ polynomials $p_i$ of algebraic degree $s$ and for $r+1$ linearized polynomials $\ell_i$ such that

$$S(x) = \sum_{i=1}^{t} p_i\big(q_i(x)\big) + \sum_{i=1}^{r} \ell_i\big(g_i(x)\big) + \ell_0(x) \ .$$

- The new method amounts to solve the linear system:

$$\begin{cases} \sum_{i=1}^{t} p_i\big(q_i(e_1)\big) + \sum_{i=1}^{r} \ell_i\big(g_i(e_1)\big) + \ell_0(e_1) & = & S(e_1) \\ \sum_{i=1}^{t} p_i\big(q_i(e_2)\big) + \sum_{i=1}^{r} \ell_i\big(g_i(e_2)\big) + \ell_0(e_2) & = & S(e_2) \\ \vdots & & \\ \sum_{i=1}^{t} p_i\big(q_i(e_{2^n})\big) + \sum_{i=1}^{r} \ell_i\big(g_i(e_{2^n})\big) + \ell_0(x) & = & S(e_{2^n}) \end{cases}$$

with (around) $t \times \frac{n^d}{s^d} + (r+1)n$ unknowns and $2^n$ equations.

- Necessary condition:

$$t \times \frac{n^d}{s^d} + (r+1)n \geqslant 2^n \quad .$$

- In practice, the lower bound was not achieved.

| | $n=4$ | $n=5$ | $n=6$ | $n=7$ | $n=8$ |
|---|---|---|---|---|---|
| $s=2$ (achieved) | **3** | **4** | **5** | **8** | **11** |
| $s=2$ (bound) | 2 | 4 | 5 | 6 | 9 |
| $s=3$ (achieved) | **2** | **3** | **3** | **4** | **4** |
| $s=3$ (bound) | 2 | 2 | 3 | 3 | 4 |

# Conclusions

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.

# Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.
- Many open issues...

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.
- Many open issues...
    - Improve proof techniques (automatize them?)

## Conclusions

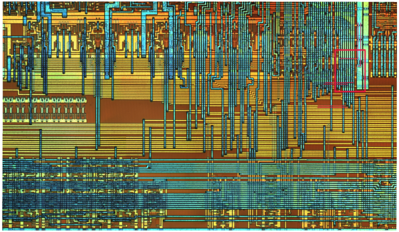
- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.
- Many open issues...
  - Improve proof techniques (automatize them?)
  - Improve existing techniques / adapt them to the SCA context

# Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.
- Many open issues...
  - Improve proof techniques (automatize them?)
  - Improve existing techniques / adapt them to the SCA context
  - Reduce the randomness consumption of existing techniques

## Conclusions

- We need algorithmic countermeasures with formal proof of resistance.
- We need formal models fitting the physical reality of devices AND enabling relatively simple proofs.
- Countermeasures must be efficient AND resistant against powerful adversaries.
- Links with many other rich fields: ECC, MPC, efficient processing in short characteristic, etc.
- Many open issues...
  - Improve proof techniques (automatize them?)
  - Improve existing techniques / adapt them to the SCA context
  - Reduce the randomness consumption of existing techniques
  - Find Efficient Evaluation methods
  - ...



Thank you for your attention!
Questions/Remarks?