# High-Order Masking by Using Coding Theory and its Application to AES

Guilhem Castagnos[1], Soline Renner[1,2], and Gilles Zémor[1]

[1] Institut de Mathématiques de Bordeaux UMR 5251, Université Bordeaux 1
351, cours de la Libération, 33405 Talence cedex, France
{guilhem.castagnos,gilles.zemor}@math.u-bordeaux1.fr
[2] Oberthur Technologies Security Group
4, allée du Doyen Georges Brus, 33600 Pessac, France
s.renner@oberthur.com

**Abstract.** To guarantee that some implementation of a cryptographic scheme is secure against side channel analysis, one needs to formally prove its leakage resilience. A relatively recent trend is to apply methods pertaining to the field of Multi-Party Computation: in particular this means applying secret sharing techniques to design masking countermeasures. It is known besides that there is a strong connection between secret sharing schemes and error-correcting codes, namely every linear code gives rise to a linear secret sharing scheme. However, the schemes mostly used in practice are the so-called Boolean masking and Shamir's secret sharing scheme and it is widely thought that they are the most adapted to masking techniques because they correspond to MDS codes that are in some sense optimal. We propose alternative masking techniques that rely on *non-MDS* linear codes: these codes are non-binary but have an underlying binary structure which is that of a *self-orthogonal binary* code. Their being non-MDS is compensated by the fact that the distributed multiplication procedure is more efficient than with MDS codes due to an efficient encoding process and that the distributed computation of squares comes at almost no cost. In protecting AES against high-order side channel analysis, this approach is more efficient than methods using Shamir's secret sharing scheme and competitive with Boolean masking.

**Keywords:** High-Order Side Channel Analysis, Linear Secret Sharing Scheme, Self-Dual Codes.

## 1 Introduction

In the 90's, Kocher *et al.* published the so-called *Side Channel Analysis* (SCA for short) which generated a huge interest in both academic and industrial communities. Indeed, they noticed that side channel leakage of an embedded device such as its power consumption or its electromagnetic radiation can reveal information on any value manipulated ([Koc96,KJJ99]). When applied during the execution of a cryptographic algorithm, such an attack can be used for secret key recovery. Since then, a wide variety of attacks has been proposed

including $t^{th}$-*order SCA* which exploits leakage observations resulting from the handling of $t$ intermediate variables during the cryptosystem processing (see *e.g.*, [Mes00,JPS05,PR10,FMPR10]).

One standard way to thwart this kind of attack is based on secret sharing and is called $t^{th}$-*order masking* when each sensitive variable is split into numerous shares in a way such that $t$ of them give no information on this variable. These shares must then be propagated independently throughout the algorithm to ensure its resistance against $t^{th}$-order SCA. In particular, when applying a *linear* secret sharing for a masked implementation of the AES cipher, shares can be easily propagated through all *linear* operations. However, much more work is needed to deal with the inversion in the finite field $\mathbf{F}_{2^8}$ involved in the AES Sbox. When masking AES, this inversion is usually (see *e.g.*, [TDG02,BMK04,OMPR05]) computed using exponentiation so masking this operation comes down to masking multiplications of sensitive variables. This last problem has been extensively addressed in the secure Multi-Party Computation literature. For instance in [BOGW88,CCD88], the authors have introduced a secure multiplication procedure for the secret sharing scheme of Shamir ([Sha79]). Much later, in [ISW03], Ishai, Sahai and Wagner have proposed another procedure applied for a basic secret sharing scheme, commonly used to design countermeasures, the so-called *Boolean masking*. These two methods have been used in the past few years to propose high-order masking schemes for AES ([RP10,KHL11,CPRR13] with Boolean Masking, and [GM11,PR11,CPR12] with Shamir's secret sharing scheme).

Let us mention that, quite recently, the authors of [BFGV12] have suggested an adaptation of the nonlinear masking technique described in [DF12] to design an implementation of AES resistant against high-order SCA. In particular, they improved the secure multiplication of [DF12]. However this scheme being nonlinear, the implementation of linear operations becomes expensive compared to linear schemes.

Another idea is to take advantage of the fact that every linear code gives rise to a linear secret sharing scheme as described in [Mas93] for example. In practice, MDS codes, such as the parity check code (corresponding to Boolean masking) and Reed-Solomon code (for Shamir's secret sharing scheme) are generally used to design countermeasures as $t^{th}$-order SCA resistance is achieved with only $t+1$ shares. Moreover, the initial Multi-Party Computation protocol proposed in [BOGW88,CCD88] for Shamir's secret sharing scheme has been generalized by Cramer *et al.* ([CDM00]) from any linear secret sharing scheme and family of codes such as *self-dual* and *geometric* codes have been suggested to improve performance of the distributed multiplication procedure ([CC06,CDG$^+$08]).

**Our Contribution.** In this paper, we propose to take advantage of results of coding theory and Multi-Party Computation to design new $t^{th}$-*order masking* techniques by selecting linear codes adapted to the masked operations. More precisely, we suggest to use *self-dual codes with a binary basis* to create secret sharing schemes in which secrets and shares belong to an extension field $K$ of

the binary field $\mathbf{F}_2$. These codes will provide low-cost square operations[3] and an efficient encoding which is extensively used as a subroutine of the secure multiplication procedure. Encoding require *zero multiplication* over the large field, contrary to all strategies based on Shamir's secret sharing scheme. As a result, our secure multiplication procedure needs $\mathcal{O}(t)$ multiplications whereas the methods based on Shamir's secret sharing scheme ([GM11,PR11]) involve $\mathcal{O}(t^3)$ multiplications[4], due to a costly encoding procedure, and the method given in [RP10], using Boolean masking, has complexity $\mathcal{O}(t^2)$ multiplications. Moreover, we propose an improvement of the multiplication procedure given in [BOGW88,CCD88,CDM00] which can also be applied for Shamir's secret sharing scheme.

Our codes are non-MDS, so we need more than $t+1$ shares to achieve $t^{th}$-order SCA resistance. However, thanks to the underlying binary structure of our code, we show that it is possible to efficiently switch code to the same code used for Boolean masking during linear operations. As a result, these linear operations can be masked as efficiently as with Boolean masking. We also propose to work with an underlying self-dual code over $\mathbf{F}_4$ which provides a low-cost $x \mapsto x^4$ operation over $K$ with less shares.

Finally, in the context of an implementation of a $t^{th}$-order secure AES, we show that our masking proposal dramatically improves the method with Shamir's secret sharing scheme and is competitive with Boolean masking.

**Paper Organization.** In Section 2, we recall the connection between $t^{th}$-order SCA countermeasures and secret sharing schemes and present the approach with Shamir's secret sharing scheme. In Section 3, we present the construction of linear secret sharing schemes derived from linear codes and the general Multi-Party Computation multiplication procedure of [CDM00] that generalizes [BOGW88,CCD88]. Section 4 is the core of our proposal: we present a new $t^{th}$-*order masking* technique based on self dual codes, and several implementation improvements. In Section 5, we apply our technique to secure an AES implementation present experimental results and make a comparison with previous works. Finally Section 5 concludes the paper.

## 2 Secret Sharing Scheme and $t^{th}$-Order Masking

An implementation of a secret key algorithm is said to be $t^{th}$-order secure if an adversary gains no information about the secret key from the knowledge of $t$ intermediate values. This ensures that the observation of the physical leakage related to the manipulation of these intermediate values will not help the adversary in performing a key recovery attack. A sound approach to reach this level of

[3] We note that a similar trick have been proposed to make efficient squaring in the long version of [PR11] but only with Reed-Solomon codes which are different to the codes used in our proposal.

[4] This can be improved to $\mathcal{O}(t^2 \log^4 t)$ multiplications by using discrete Fourier transform cf. [CPR12].

security is to mask each sensitive variable $s$ with a linear secret sharing scheme. For example, the so-called Boolean masking consists in randomly splitting $s$ into $t+1$ shares $s_1, \ldots, s_{t+1}$ in a way such that $s = s_1 + \cdots + s_{t+1}$ ([RP10,CPRR13]). The linear secret sharing scheme of Shamir has also been used in this context ([GM11,PR11,CPR12]).

Some mechanisms must be developed to make this masking procedure compatible with the operations performed in the protected algorithm, *i.e.*, to enable the computation on masked data. For example, for the AES cipher, linear operations are compatible with linear secret sharing. Nonlinear functions involved in the *SubBytes* transformation are more difficult to deal with. Usually the inversion involved in this transformation is computed using an exponentiation which requires a method to protect multiplications of sensitive variables. In the context of the AES, some solutions have been developed in [RP10,KHL11,CPRR13] for multiplication with Boolean masking.

In this section, after some definitions on secret sharing schemes, we describe the solution, close to our proposal, given in [GM11,PR11,CPR12] still in the context of AES, to perform multiplications with Shamir's secret sharing scheme.

### 2.1 Definitions

Let $K$ be a field. A secret sharing scheme is a method to split a secret $s \in K$ among a set of $n$ shares. More precisely a secret sharing scheme is composed of two algorithms, *encoding* and *decoding* . The *encoding* of $s$ provides an $n-$vector of shares called *share vector*: $(s_1, \ldots, s_n) \in K^n$. The *decoding* algorithm reconstructs the secret $s$ from $(s_1, \ldots, s_n)$.

A secret sharing scheme has *$t$-privacy* if any set of at most $t$ shares reveals no information about the secret, and *$r$-reconstruction* if $r$ shares reveal the entire secret. A secret sharing scheme is said to be *linear* if for any two secrets $s$ and $s'$ shared respectively by $(s_1, \ldots s_n)$ and $(s'_1, \ldots s'_n)$, the vectors $(s_1+s'_1, \ldots, s_n+s'_n)$ and $(\lambda s_1, \ldots, \lambda s_n)$ decode respectively to $s + s'$ and $\lambda s$, $\lambda \in K$.

A $t^{th}$-order secure implementation of an algorithm can be based on a linear secret sharing scheme with $t-$privacy. The *encoding* procedure is applied to each input variable. Share vectors are then manipulated to reflect the protected algorithm. Additions and scalar multiplications of secrets are performed easily on the share vectors. In the following, we recall the linear secret sharing scheme of Shamir and the method to perform $t^{th}$-order secure multiplication of secrets described in [GM11,PR11,CPR12].

### 2.2 Shamir's Secret Sharing Scheme

In [Sha79], Shamir has introduced a linear secret sharing scheme over a finite field $\mathbf{F}_q$ based on polynomial interpolation. The *encoding* step consists in generating a random secret polynomial $P$ of degree $t$ over $\mathbf{F}_q$ such that $P(0) = s$. Then, the share vector of $s$ denoted $(s_1, \ldots, s_n)$ is generated by evaluating $s_i = P(x_i)$ in $n$ distinct non-zero points $x_1, \ldots, x_n$ of $\mathbf{F}_q$. Let $A$ be a subset of $\{1, \ldots, n\}$ such that $|A| \geqslant t + 1$. The *decoding* step consists in recovering the secret polynomial

by interpolation and then the secret by an evaluation at 0. This is done by computing:

$$s = \sum_{i \in A} s_i \beta_i(0), \tag{1}$$

where $\beta_i(X) = \prod_{j \in A, j \neq i} \frac{X - x_j}{x_i - x_j}$ are Lagrange polynomials. This gives a linear secret sharing scheme with $t$-privacy and $t+1$-reconstruction.

Linear operations on secrets can be securely executed as described in subsection 2.1. However nonlinear operations over $\mathbf{F}_q$ are more complex to deal with. Multiplication of two secrets has been extensively studied to design secure multiparty computation schemes, *e.g.* in [BOGW88,CCD88,GRR98]. The method consists in multiplying the two share vectors share by share. This operation corresponds to the multiplication of two degree $t$ polynomials which gives a polynomial of degree $2t$. The secret result of the multiplication can then be recovered with at least $2t+1$ shares if $n \geqslant 2t+1$. A method to reduce the number of shares required to reconstruct the secret is needed, otherwise $k$ successive multiplications would require to take $n \geqslant kt+1$. The solution consists in *re-encoding* $2t+1$ shares and to compute the sum of the resulting share vectors.

In [GM11,PR11], this secure multiplication has been used to implement the AES cipher. Algorithm 1 recalls the solution to perform the secure multiplication procedure as given in [GM11]. In particular the authors have suggested to take $n = t+1$ during the whole process and to generate on-the-fly the additional shares when a multiplication step is required reducing the overall complexity of their AES implementation.

**Algorithm 1** Secure Multiplication [GM11, Algorithm 2]

INPUTS: $2n-1$ distinct non-zero public elements $x_1, \ldots, x_{2n-1}$
$(s_1, \ldots, s_n)$ a share vector of $s$ and $(s'_1, \ldots, s'_n)$ a share vector of $s'$
$\beta_j(x_i)$ pre-computed for $1 \leqslant j \leqslant n$ and $n+1 \leqslant i \leqslant 2n-1$
$\beta_i^*$ pre-computed for $1 \leqslant i \leqslant 2n-1$ with $\beta_i^* = \prod_{j=1, j \neq i}^{2n-1} \frac{-x_j}{x_i - x_j}$
OUTPUT: $(z_1, \ldots, z_n)$ a share vector of $ss'$

1. **For** $i = n+1$ **to** $2n-1$ **do**
2. $\quad s_i \leftarrow \sum_{j=1}^{n} s_j \beta_j(x_i)$
3. $\quad s'_i \leftarrow \sum_{j=1}^{n} s'_j \beta_j(x_i)$
4. **For** $i = 1$ **to** $2n-1$ **do**
5. $\quad w_i = s_i s'_i \beta_i^*$
6. $\quad (w_{i_1}, \ldots, w_{i_n}) \leftarrow encoding(w_i)$
7. **For** $j = 1$ **to** $n$ **do**
8. $\quad z_j \leftarrow \sum_{i=1}^{2n-1} w_{i_j}$
9. **Return** $(z_1, \ldots, z_n)$

*Remark 1.* When the field $\mathbf{F}_q$ has characteristic 2, the square of a secret can be performed more efficiently than general multiplication. Let us consider the square $(s_1^2, \ldots, s_n^2)$ of a share vector of $s$ and let us denote $P$ the secret polynomial used to share $s$. Relation (1) gives $s^2 = \sum_{i \in A} s_i^2 \beta_i(0)^2$, so the secret $s^2$ can be

recovered with only $(t+1)$ shares. However each share $s_i^2$ is an evaluation of $P^2$ at $x_i^2$ instead of $x_i$. In [GM11] the authors propose again to apply a re-*encoding* step so that the shares are the evaluation of a polynomial at $(x_1, \ldots, x_n)$. In the long version of [PR11] it is proposed to choose the points $x_1, \ldots, x_n$ such that the set of these points is stable with respect to the Frobenius map $x \mapsto x^2$. As a result, a simple re-ordering of the shares is needed instead of a re-*encoding* step. The situation will be even simpler with our proposal since no re-ordering will be needed.

The main cost of Algorithm 1 comes from the numerous multiplications by constant values, namely the multiplications by $\beta_i^*$ and $\beta_j(x_i)$ and the ones required during the *encoding* steps. These steps correspond to evaluations of polynomials at $n$ different points. In [CPR12], the authors propose to use the discrete Fourier transform for these computations. Hence the whole secure multiplication procedure can be improved to a complexity of $\mathcal{O}(n^2)$ multiplications instead of the naive approach in $\mathcal{O}(n^3)$ multiplications. With our proposal, no multiplication by constant values over $\mathbf{F}_q$ will be needed, as a result of which the secure multiplication procedure will be improved to a complexity of $\mathcal{O}(n)$ multiplications.

## 3 Coding Theory Generalisation

It is well-known that a linear secret sharing scheme can be built from a linear code as described, for example, in [Mas93,CC06,CCG+07,CDG+08]. Moreover, the problem of computing on masked data (addition and multiplication of secret values) for general secret sharing schemes has been addressed to design secure multiparty computation protocol (see, *e.g.*, [CDM00]), generalizing the protocol initially proposed in [BOGW88,CCD88] with Shamir's scheme. In the following, we recall some basic definitions, explain the construction of a linear secret sharing scheme from a linear code and describe the algorithms to perform secure computation on secrets that generalize the algorithms given in the previous section.

### 3.1 Basic Definitions and Results from Coding Theory

Over a finite field $\mathbf{F}_q$, an $[n+1, k+1, d]_q$ linear code $\mathcal{C}$ is a $(k+1)-$dimensional vector subspace of $\mathbf{F}_q^{n+1}$ with minimum Hamming distance $d$. The generator matrix $G$ of a linear code in systematic form can be written as $G = [Id_{k+1} |\ \mathcal{A}\ ]$, where $\mathcal{A}$ is a $(k+1)\times(n-k)-$matrix. The elements $c = (c_0, \ldots, c_n)$ of $\mathcal{C}$ are called codewords and can be generated as $c = (r_0, \ldots, r_k)\cdot G$, where $(r_0, \ldots, r_k) \in \mathbf{F}_q^{k+1}$. The dual code $\mathcal{C}^\perp$ of $\mathcal{C}$ is an $[n+1, n-k]_q$ linear code defined by

$$\mathcal{C}^\perp = \left\{ c \in \mathbf{F}_q^{n+1} : \langle c, c' \rangle = 0 \text{ for all } c' \in \mathcal{C} \right\} ,$$

where $\langle ., . \rangle$ denotes the *inner product* defined by $\langle c, c' \rangle = \sum_{i=0}^{n} c_i c'_i$. When $\mathcal{C} = \mathcal{C}^\perp$ (resp. $\mathcal{C} \subseteq \mathcal{C}^\perp$), $\mathcal{C}$ is said to be *self-dual* (resp. *self-orthogonal*).

From $\mathcal{C}$ an $[n+1, k+1]_q$ linear code, the *squared code* denoted $\widehat{\mathcal{C}}$ is defined by

$$\widehat{\mathcal{C}} = \langle \{c * c',\ c, c' \in \mathcal{C}\} \rangle \ ,$$

where $*$ denotes the *Schur product* $c * c' = (c_0 c'_0, \ldots, c_n c'_n)$. Moreover we define $\mathcal{C}^2$ as

$$\mathcal{C}^2 = \left\langle \{c^2 = c * c,\ c \in \mathcal{C}\} \right\rangle .$$

For more details on linear codes, the interested reader may refer to [MS78]. In the following we give two lemmas useful to select suitable codes for efficient operations on masked data.

**Lemma 1.** *Let $\mathcal{C}$ be an $[n, k]$ linear code over $\mathbf{F}_q$ of characteristic 2. The following assertions are equivalent:*

1. $\forall c \in \mathcal{C}$, $c^2 \in \mathcal{C}$,
2. $\mathcal{C}$ *has a binary basis.*

*Proof.* Let $\mathcal{C}$ be a linear code over $\mathbf{F}_q$ of characteristic 2 having a binary basis $(b_1, \ldots, b_k)$ with $b_i \in \mathbf{F}_2^n$. For all $i$, $b_i^2 = b_i$. If $c$ is a codeword of $\mathcal{C}$, then there exists a linear combination such that $c = \sum_{i=1}^{k} \lambda_i b_i$ with $\lambda_i \in \mathbf{F}_q$. We have $c^2 = \sum_{i=1}^{k} \lambda_i^2 b_i^2 = \sum_{i=1}^{k} \lambda_i^2 b_i \in \mathcal{C}$.

Conversely, let $\mathcal{C}$ be a $[n, k]$ linear code over $\mathbf{F}_q$, such that for all codeword $c \in \mathcal{C}, c^2 \in \mathcal{C}$. Let $G$ be a generator matrix of $\mathcal{C}$ in systematic form and $b_i$ be the $i^{th}$ row of $G$. Let $a_1, \ldots, a_{n-k}$ be elements of $\mathbf{F}_q$ such that $b_i = (0, \ldots, 0, 1, 0, \ldots, 0, a_1, \ldots, a_{n-k})$.

By definition $b_i^2 = (0, \ldots, 0, 1, 0, \ldots, 0, a_1^2, \ldots, a_{n-k}^2) \in \mathcal{C}$, so there exists a linear combination such that $b_i^2 = \sum_{i=1}^{k} \lambda_i b_i$. From the last equality, the $j^{th}$ coordinate of $b_i^2$ is equal to $\lambda_j$ for $j \in \{1, \ldots, k\}$. By identification, we have for $1 \leqslant j \leqslant k$ and $j \neq i$, $\lambda_j = 0$, and $\lambda_i = 1$. Therefore $b_i^2 = b_i$ and then $a_j^2 = a_j$ for $j \in \{1, \ldots, n-k\}$. Thus $b_i \in \mathbf{F}_2^n$. □

**Lemma 2.** *If $\mathcal{C}$ is a linear self-dual code (or a linear self-orthogonal code), then the codeword $\mathbf{1} = (1, \ldots, 1) \in \widehat{\mathcal{C}}^{\perp}$.*

*Proof.* For all $c, c' \in \mathcal{C}$ of the self-dual (or self-orthogonal) code, we have: $\langle c, c' \rangle = \langle c * c', \mathbf{1} \rangle = 0$. □

### 3.2 Construction of a Linear Secret Sharing Scheme from a Linear Code

Let $\mathcal{C}$ be an $[n+1, k+1, d]_q$ linear code with $G$ its generator matrix in systematic form. All codewords $c = (c_0, c_1, \ldots, c_n)$ of $\mathcal{C}$ can be identified with a share vector $(c_1, \ldots, c_n)$ of the secret $c_0 = s$. Hence the *encoding* procedure of a secret $s \in \mathbf{F}_q$ consists in generating a codeword $c = (s, r_1, \ldots, r_k) \cdot G$, where $r_1, \ldots, r_k$ are random values of $\mathbf{F}_q$. In case of ambiguity, this procedure is denoted $encoding_{\mathcal{C}}$. Assuming that there exists a codeword $h = (h_0, \ldots, h_n)$ of $\mathcal{C}^{\perp}$ such that $h_0 = 1$,

the *decoding* procedure can be implemented by computing $s = \sum_{i=1}^{n} \lambda_i c_i$ where $\lambda_i = -h_i \in \mathbf{F}_q$. In this case, we call *recombination vector* of $\mathcal{C}$, such a vector $\lambda = (\lambda_1, \ldots, \lambda_n)$ where the number of non-zero $\lambda_i$ equals to $d^\perp - 1$.

In [CCG+07, Theorem 1], it is shown that such a linear secret sharing scheme has $(d^\perp - 2)-$privacy and $(n - d + 2)-$reconstruction, where $d^\perp$ is the minimum distance of $\mathcal{C}^\perp$.

### 3.3 Operations on Masked Data

A linear code gives a linear secret sharing scheme, so addition and scalar multiplication of secrets correspond to addition and scalar multiplication of share vectors. Consider now the problem of multiplying two secrets shared by a general secret sharing scheme. In other words, from the shared vectors of two secrets one wishes to obtain a shared vector representing the product of the secrets by using operations on shares and without reconstructing the secrets. This problem has been addressed in [CDM00]. In this work, a procedure that generalizes Algorithm 1 is given, by considering any linear code $\mathcal{C}$ such that $\widehat{d}^\perp \geq d^\perp$ where $\widehat{d}^\perp$ is the minimum distance of $\widehat{\mathcal{C}}$. We describe this approach below.

We assume that there exists a recombination vector $\widehat{\lambda}$ of $\widehat{\mathcal{C}}$ (for example, according to Lemma 2, we can choose $\widehat{\lambda} = (1, \ldots, 1)$ if $\mathcal{C}$ is self-dual or self-orthogonal). Let $c, c' \in \mathcal{C}$ be such that $c_0 = s$ and $c'_0 = s'$. The secret multiplication $ss'$ can be shared by $c * c' \in \widehat{\mathcal{C}}$ and recovered by computing $ss' = \sum_{i=1}^{n} \widehat{\lambda}_i c_i c'_i$. Furthermore $\widehat{\mathcal{C}}$ gives a secret sharing scheme with $(\widehat{d}^\perp - 2)-$privacy and as $\widehat{d}^\perp \geq d^\perp$, the privacy level of the secret sharing scheme associated to $\mathcal{C}$ is preserved. To be able to perform further multiplications, we need a method for *re-encoding* the codeword $c * c' \in \widehat{\mathcal{C}}$ into a new codeword $z \in \mathcal{C}$ such that $z_0 = ss'$. As seen in the previous section, this can be done by *re-encoding* each $\widehat{\lambda}_i c_i c'_i$ into a new codeword of $\mathcal{C}$. Then by summing these $n$ codewords, we obtain $z$, a codeword of $\mathcal{C}$, such that $z_0 = ss'$. This procedure is described in the following algorithms: Algorithm 3 for the secure multiplication procedure, and Algorithm 2 for the *re-encoding* subroutine.

**Algorithm 2** Secure Re-*encoding* Procedure

INPUTS: $\mathcal{C}$ a $[n+1, k+1]_q$ linear code
$\mathcal{C}'$ a $[n'+1, k'+1]_q$ linear code and $\lambda'$ a recombination vector
$(w_1, \ldots, w_{n'})$ a share vector corresponding to a codeword of $\mathcal{C}'$

OUTPUT: $(z_1, \ldots, z_n)$ a share vector of $\sum_{i=1}^{n'} \lambda'_i c_i$ corresponding to a codeword of $\mathcal{C}$

**Function:** *re-encoding* $_{\mathcal{C}' \mapsto \mathcal{C}}(\lambda', w_1, \ldots, w_{n'})$

1. **For** $i = 1$ **to** $n'$ **do**
2. $\quad (w_{i_1}, \ldots, w_{i_n}) \leftarrow encoding_{\mathcal{C}}(\lambda'_i w_i)$
3. **For** $j = 1$ **to** $n$ **do**
4. $\quad z_j \leftarrow \sum_{i=1}^{n'} w_{i_j}$
5. **Return** $(z_1, \ldots, z_n)$

**Algorithm 3** Secure Multiplication Procedure

INPUTS: $\mathcal{C}$ a $[n+1, k+1]_q$ linear code
$(c_1, \ldots, c_n)$ a share vector of $s$ and $(c'_1, \ldots, c'_n)$ a share vector of $s'$ corresponding to codewords of $\mathcal{C}$
$\widehat{\lambda}$ a recombination vector of $\widehat{\mathcal{C}}$
OUTPUT: $(z_1, \ldots, z_n)$ a share vector of $ss'$ corresponding to a codeword of $\mathcal{C}$

1. $(w_1, \ldots, w_n) \leftarrow (c_1 c'_1, \ldots, c_n c'_n)$
2. $(z_1, \ldots, z_n) \leftarrow$ *re-encoding* $_{\widehat{\mathcal{C}} \mapsto \mathcal{C}}(\widehat{\lambda}, w_1, \ldots, w_n)$
3. **Return** $(z_1, \ldots, z_n)$

These algorithms requires $n$ multiplications of shares and numerous multiplications by constant values: the coordinates of $\widehat{\lambda}$ and the elements of the matrix $G$ for the encodings. In our proposal, all these elements will be binary, so only $n$ multiplications will be needed for a secure multiplication.

**Fact 1** *From the properties of the secure Multi-Party Computation protocol given in [CDM00, Section 6], Algorithms 2 and 3 give a $t^{th}$-order secure multiplication, with $t = d^{\perp} - 2$ where $d^{\perp}$ is the dual distance of the linear code $\mathcal{C}$.*

*Remark 2.* Over a finite field $\mathbf{F}_q$ of characteristic 2, the square of a secret $s$ can be computed without using the *squared code* $\widehat{\mathcal{C}}$ and $\widehat{\lambda}$. Indeed, if $(1, \lambda_1, \ldots, \lambda_n) \in \mathcal{C}^{\perp}$, then $(1, \lambda_1^2, \ldots, \lambda_n^2) \in \mathcal{C}^{2^{\perp}}$. As a result, if $s$ is shared by $(c_1, \ldots, c_n)$, one can apply Algorithm 2 to $\lambda^2, (c_1^2, \ldots, c_n^2)$, to obtain a share vector of $s^2$ corresponding to a codeword of $\mathcal{C}$. Moreover, no re-encoding is needed if $c^2 \in \mathcal{C}$. According to Lemma 1, this will be the case if and only if $\mathcal{C}$ has a binary basis. In this case, the secure squaring procedure only requires to compute $n$ squares.

### 3.4 Application to Boolean Masking and to Shamir's Secret Sharing Scheme

The well-known Boolean masking can be obtained with this framework by using the $[n+1, n]_q$ linear parity check code with generator matrix

$$G = \left( \begin{array}{c|c} & 1 \\ Id_n & \vdots \\ & 1 \end{array} \right). \tag{2}$$

The dual of this code is generated by the codeword $(1, \ldots, 1)$. As a consequence, the secret sharing scheme constructed from such a code has $(n-1)$-privacy and the secret $s \in \mathbf{F}_q$ can be recovered by computing $s = \sum_{i=1}^{n} c_i$, where $(c_1, \ldots, c_n)$ is a share vector of $s$. As said in Remark 2 this scheme allows a secure squaring procedure with a low computational cost. However for $n + 1 \geqslant 2$, it is easy to see that $\widehat{\mathcal{C}}^{\perp} = \{0\}$. As a consequence we cannot apply Algorithm 3 to

perform a secure multiplication. Nevertheless other methods are proposed in the literature to perform such an operation without using the framework of error-correcting codes (for instance the method given in [RP10] requires roughly $n^2$ multiplications of shares).

Shamir's secret sharing scheme can be constructed from the Reed-Solomon code of parameters $[n+1, t+1]_q$ with the generator matrix

$$G = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & x_1 & \dots & x_n \\ 0 & x_1^2 & \dots & x_n^2 \\ \vdots & \vdots & \dots & \vdots \\ 0 & x_1^t & \dots & x_n^t \end{pmatrix},$$

with for all $i, j$, $i \neq j$, $x_i \neq x_j \in \mathbf{F}_q \setminus \{0\}$. The recombination vector $\lambda$ can be chosen such that $\lambda_i = \beta_i(0)$ (*i.e.*, Lagrange polynomials evaluated in 0). Moreover, $\widehat{\mathcal{C}}$ is the $[n+1, 2t+1]_q$ Reed-Solomon code and if $2t+1 \leqslant n$, then we can apply Algorithm 3. This gives essentially Algorithm 1 where the only difference is the on-the-fly computation of the missing shares. Similarly the method of [GM11] to compute secure squaring for Shamir's scheme explained in Remark 1, corresponds to the method given in Remark 2.

## 4 Our Contribution

The codes generally considered for the construction of secret sharing schemes are MDS codes, such as the parity check code (for Boolean masking) and Reed-Solomon codes (for Shamir's scheme). This is because they give *perfect* secret sharing schemes, namely $t-$privacy and $t+1-$reconstruction. In particular, when used as $t^{th}$-order masking, only $t+1$ shares can be used to mask each input variable.

The efficiency of Boolean masking comes from the fact that it corresponds to a code with a *binary* generator matrix (2). By lemma 1, this binary basis implies that squaring in an extension of the binary field is a low-cost operation. Moreover, *encoding* requires only additions and no multiplications. As mentioned in subsection 3.4, the general secure multiplication (Algorithm 3) cannot be applied to Boolean masking, and the secure multiplication algorithm of [RP10] requires $\mathcal{O}(t^2)$ multiplications. For Shamir's secret sharing scheme [PR11,GM11], secure multiplication corresponds to Algorithm 3. But the *encoding* subroutine needs numerous multiplications in the finite field and secure multiplication has complexity $\mathcal{O}(t^3)$ or $\tilde{\mathcal{O}}(t^2)$ multiplications with FFT techniques ([CPR12]), even if only $\mathcal{O}(t)$ multiplications of *shares* are needed. Moreover, some work has been done to make the squaring procedure more efficient (cf. remark 1).

We propose to select a family of non-MDS linear codes over an extension of the binary field such that Algorithm 3 is applicable like in Shamir's scheme, and such that a binary basis is available like in Boolean masking. With these codes we will have *all the benefits*: an *encoding* subroutine which requires zero

multiplication, a low-cost square operation (like in Boolean masking), and a secure multiplication procedure which requires only $\mathcal{O}(t)$ multiplications and is thus more efficient than methods with MDS codes. The codes that we use have an underlying binary structure which is that of a self-dual or a self-orthogonal binary code. We also discuss the case of codes defined with a basis in $\mathbf{F}_4$.

Moreover, we propose an improvement of the multiplication procedure described in Algorithm 3 which can also be applied for Shamir's secret sharing scheme.

Finally, to compensate the additional number of shares needed by the fact that our codes are non-MDS, we show that it is possible to efficiently switch code to the same code used for Boolean masking during linear operations, thanks to the underlying binary structure of our codes. As a result, these linear operations can be masked as efficiently as with Boolean masking.

Table 3 sums up the costs of our masking procedure when applying all these improvements.

### 4.1 Linear Secret Sharing Schemes based on Self-Dual Codes

In all the following we consider a base field $\mathbf{F}_q$ of characteristic 2. As previously said, we want to select a linear code $\mathcal{C}$ over $\mathbf{F}_q$ with a binary basis. Moreover, as described in Section 3, to ensure that the multiplication procedure given by Algorithm 3 is applicable, $\widehat{\mathcal{C}}^{\perp}$ must contains a particular codeword $h$ such that $h_0 = 1$. Lemma 2 shows that by choosing for $\mathcal{C}$ a self-dual or self-orthogonal code, such a property is fulfilled.

As described in Section 3, a linear $[n+1, k+1]$ code allows to construct a secret sharing scheme with $n$ shares and $t-$privacy, where $t = d^{\perp} - 2$. For a fixed value $t$, the number of shares must be the smallest possible to reduce the total number of operations of a $t^{th}$-order masking. In the coding literature, self-dual and self-orthogonal binary codes are well-studied. In particular, we can give in Table 1 a list of binary codes $\mathcal{C}$ with minimal length and dual distance $d^{\perp} = t+2$ for $1 \leqslant t \leqslant 6$, such that for even length, the codes $\mathcal{C}$ are self-dual and for odd length, self-orthogonal. The code used for our $t^{th}$-order masking is not strictly speaking the code $\mathcal{C}$ but the code over $\mathbf{F}_q$ generated by the binary generator matrix of $\mathcal{C}$. The code constructed over $\mathbf{F}_q$ remains self-dual or self-orthogonal, and has the same properties (length, dimension, minimum distance and dual distance) than the underlying binary code.

In this table, the self-dual shortened Golay code [22,11,6] is built from the extended Golay code by using codewords beginning by 00 and 11. The code $C_{21}$ [21,11,5] is obtained by removing a coordinate of the shortened Golay code. For larger values of $t$, the reader is referred to the codes given in [CS90,GO03], and it is known that $n$ will be linear in $t$.

Interestingly, self-dual binary codes and some generalisations have been called upon [CGKS12,CG13] in order to improve resistance against side-channel analysis, through in contexts that do not invoke secret sharing.

To construct a $t^{th}$-order masking scheme, we select the $[n+1, k+1]$ code from the line $t$ of Table 1. The *encoding* procedure (cf. subsection 3.2) requires the

| $t$ | Binary code $\mathcal{C}$ $[n+1, k+1]$ | Binary Dual code $\mathcal{C}^{\perp}$ with $d^{\perp} = t+2$ | Number of additions required during an *encoding* |
|---|---|---|---|
| 1 | Code $[7, 3]$ | Hamming code $[7, 4, 3]$ | 5 |
| 2 | Extended Hamming code $[8, 4, 4]$ | | 8 |
| 3 | Code $[21, 10]$ | $C_{21}$ $[21, 11, 5]$ | 48 |
| 4 | Shortened Golay code $[22, 11, 6]$ | | 44 |
| 5 | Code $[23, 11]$ | Golay code $[23, 12, 7]$ | 64 |
| 6 | Extended Golay code $[24, 12, 8]$ | | 72 |

**Table 1.** List of binary codes

generation of $k$ random values over $\mathbf{F}_q$ and only $L$ additions, where $L$ is given in Table 1 and depends on the number of 1s in the generator matrix considered. For this *encoding* step, there is no multiplication by constant values of $\mathbf{F}_q$ unlike in Shamir's scheme. The addition and scalar multiplication of a secret are computed on each of the $n$ shares, so $n$ operations are needed. Squaring also consists in squaring the $n$ shares. Secure multiplication is done with Algorithm 3. The vector $\widehat{\lambda}$ is defined over $\mathbf{F}_2$, so only $n$ multiplications in $\mathbf{F}_q$ are required.

From Table 1, we note that the number of shares $n$ of our $t^{th}$-order masking scheme is important compared to a perfect scheme such as Shamir's. For example for $t = 3$, we need $n = 20$ shares and with a perfect masking only 4. To improve the performance of our masking method, a solution is to consider an underling self-dual or self-orthogonal code over $\mathbf{F}_4$ instead of $\mathbf{F}_2$ if $\mathbf{F}_4 \subset \mathbf{F}_q$. Indeed, as we can see in [GO03] such codes provide a better ratio $n/t$. With such a code the generator matrix has now its coefficients in $\mathbf{F}_4 = \{0, 1, w, w+1\} \subset \mathbf{F}_q$. As a result the *encoding* procedure requires some multiplications by $w$. However, unlike in Shamir's scheme, here only one constant value, *i.e.*, $w$ is manipulated and the products $wx$, $x \in \mathbf{F}_q$ can be precomputed in a table.

In Table 2, we give a list of optimal codes over $\mathbf{F}_4$ and indicate the number of additions and *the number of low-cost multiplications with* $w$ required during an encoding procedure. As with Table 1, the code used in our masking scheme is a self-dual or a self-orthogonal code over $\mathbf{F}_q$ built from the generator matrix of the code given in the table. With these codes, we lose the advantage of the low-cost squaring operation: the secure squaring now requires a re-encoding procedure as described in Remark 2. However the complexity of raising a sensitive variable to the power 4 is still small. Indeed by adapting Lemma 1, we can show that for any codeword $c \in \mathcal{C}$, we have $c^4 \in \mathcal{C}$.

### 4.2 Improvement of Secure Multiplication

During a secure multiplication (Algorithm 3), the most expensive step is the re-encoding process from $\widehat{\mathcal{C}}$ to $\mathcal{C}$ where $n$ calls are done to the *encoding* procedure. In order to reduce the complexity of this algorithm, we show that this number

| $t$ | Code $\mathcal{C}$ over $\mathbf{F}_4$ $[n+1, k+1]$ | Dual code $\mathcal{C}^\perp$ with $d^\perp = t+2$ | *encoding* add | mult with w |
|---|---|---|---|---|
| 1 | Extended quadratic residue code `XQR`(3) $[4, 2, 3]$ | | 4 | 2 |
| 3 | Code $[11, 5]$ | Quadratic residue code `QR`(11) $[11, 6, 5]$ | 25 | 5 |
| 4 | Extended quadratic residue code `XQR`(11) $[12, 6, 6]$ | | 32 | 6 |
| 5 | Code $[19, 9]$ | Quadratic residue code `QR`(19) $[19, 10, 7]$ | 69 | 9 |

**Table 2.** List of codes over $\mathbf{F}_4$

of calls can be decreased. This modified algorithm will still be $t^{th}$-order SCA secure with $t = d^\perp - 2$.

After the multiplication of two share vectors of $s$ and $s'$, we obtain a share vector $(c_1c'_1, \ldots, c_nc'_n)$ of $ss'$ corresponding to a codeword of $\widehat{\mathcal{C}}$. If we add the vector $(c_1c'_1, \ldots, c_nc'_n)$ and a random share vector of 0 in $\widehat{\mathcal{C}}$, then we obtain a *random*[5] share vector in $\widehat{\mathcal{C}}$ of $ss'$ denoted $w = (w_1, \ldots, w_n)$. Furthermore, $\widehat{\mathcal{C}}$ gives a secret sharing scheme with $(\widehat{d}^\perp - 2)-$privacy. Assuming that $e = \widehat{d}^\perp - d^\perp > 0$, we combine $(e+1)$ elements of $w$ giving a share vector

$$\tilde{w} = \left( \sum_{i=1}^{e+1} \widehat{\lambda}_i w_i, w_{e+2}, \ldots w_n \right)$$

associated to a linear code $\widehat{\mathcal{C}}^*$ of length $n - e$. By construction, the vector $\widehat{\lambda}^* = (1, \widehat{\lambda}_{e+2}, \ldots, \widehat{\lambda}_n)$ is a recombination vector of this code if $(\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n)$ is a recombination vector of $\widehat{\mathcal{C}}$ . The algorithm is still $t^{th}$-order SCA secure if we re-encode the coordinates of this share vector $\tilde{w}$ instead of $w$. Indeed, suppose that an adversary has access to a subset of $t$ shares of this vector $\tilde{w}$. If the first coordinate is not in this subset, the adversary has no information on the secret as the scheme is at least $t-$private. If he knows the first coordinate he has less information than with $w_1, w_2, \ldots, w_{e+1}$ and $t-1$ others shares, *i.e.*, with $\widehat{d}^\perp - 2$ shares, so he has again no information on the secret.

Therefore, only $n - e$ shares can be re-encoded during the secure multiplication procedure as described in Algorithm 4.

This improvement can be applied for each linear code with $\widehat{d}^\perp > d^\perp$. In particular, the squared codes $\widehat{\mathcal{C}}$ associated to the codes given in Tables 1 and 2 are the parity check codes of length $n+1$ and have $\widehat{d}^\perp = n+1$, so only $n-e = t+1$ shares have to be re-encoded in Step 5. Similarly, for Shamir's secret sharing scheme by taking $n = 2k+1$, then we have that $t = k$, $d^\perp = k+2$ and the squared code $\widehat{\mathcal{C}}$ associated is the Reed-Solomon code of parameters $[2k+2, 2k+1]$, so $\widehat{d^\perp} = 2k+2$. Therefore $n - e = t+1$.

[5] If this step is omitted, the vector $y = c * c'$ of $\widehat{\mathcal{C}}$ may not have $(\widehat{d}^\perp - 2)-$privacy. For example, if $s = s'$ and the two input share vectors are equals, $y \in \mathcal{C}^2$ and as $\mathcal{C}^2 = \mathcal{C}$ in our proposal, this vector has only $(d^\perp - 2)-$privacy.

**Algorithm 4** Improvement of Secure Multiplication

INPUTS: $\mathcal{C}$ a $[n+1, k+1]_q$ linear code
$(c_1, \dots c_n)$ and $(c'_1, \dots, c'_n)$ two share vectors respectively of $s$ and $s'$
$\widehat{\lambda}$ a recombination vector of $\widehat{\mathcal{C}}$ and $e = \widehat{d}^{\perp} - d^{\perp} > 0$
OUTPUT: $(z_1, \dots, z_n)$ a share vector of $ss'$

**Function:** SecMult$((c_1, \dots c_n), (c'_1, \dots c'_n))$
1. $(w_1, \dots, w_n) \leftarrow (c_1 c'_1, \dots, c_n c'_n) + encoding_{\widehat{\mathcal{C}}}(0)$
2. $(w_1, \dots, w_n) \leftarrow (\widehat{\lambda}_1 w_1, \dots, \widehat{\lambda}_{e+1} w_{e+1}, w_{e+2}, \dots, w_n)$
3. **For** $i = 1$ **to** $e$ **do**
4. $\quad w_{e+1} \leftarrow w_{e+1} + w_i$
5. $(z_1, \dots, z_n) \leftarrow$ re-$encoding_{\ \widehat{\mathcal{C}}^* \mapsto \mathcal{C}}((1, \widehat{\lambda}_{e+2}, \dots, \widehat{\lambda}_n), (w_{e+1}, \dots, w_n))$
6. **Return** $(z_1, \dots, z_n)$

### 4.3 Code Switching to Perform Efficient Linear Operations

To compensate the additional number of shares needed by the fact that our codes are non-MDS, we propose a solution to *reduce the number of shares used during linear operations*, still achieving a $t^{th}$-order masking. Let us consider the masking scheme using a code $\mathcal{C}$ built from Table 1. Thanks to the underlying binary structure, it is possible to efficiently re-encode the share vectors of $\mathcal{C}$ to an MDS code $\mathcal{C}^\star$ for linear operations (additions and scalar multiplications). This simply consists in considering only the shares involved in the reconstruction, namely the $t+1$ shares corresponding to the non-zero coordinates of the recombination vector. As a result, the MDS code $\mathcal{C}^\star$ corresponds to Boolean masking. Hence all linear operations can be implemented with *the same complexity as Boolean masking*. At the end of the linear operations, when a multiplication has to be done, each share will be re-encoded to form a new share vector corresponding to a codeword of $\mathcal{C}$.

This method can be adapted for a masking scheme using the codes $\mathcal{C}$ of Table 2. At the end of the multiplication procedure, the shares of a vector of $\widehat{\mathcal{C}}$ are re-encoded into $\mathcal{C}^\star$ (instead of $\mathcal{C}$) to form a share vector of length $t+1$. This code is used during the linear operations and then a re-encoding is applied so as to fall back on a codeword of $\mathcal{C}$ when another multiplication is needed.

### 4.4 Comparison with Other Masking Schemes

We summarize in Table 3 the cost of secure operations when we use our $t^{th}$-order masking schemes derived from the codes of Table 1 (resp. from the codes of Table 2) denoted by *our masking scheme* $\mathbf{F}_2$ (resp. *our masking scheme* $\mathbf{F}_4$) using the improvements proposed in subsections 4.2 and 4.3.

In this table, *rand* indicates the number of random elements to generate, *add* and *mult* correspond to the numbers of additions and multiplications in the finite field $\mathbf{F}_q$. By *mult with w*, we indicate the number of small multiplications with the constant $w \in \mathbf{F}_4$ which can be performed with a look-up table. We also

give the cost of masked operations for Boolean masking using the multiplication procedure described in [RP10] and for Shamir's secret sharing scheme using the multiplication procedure of [GM11,PR11]. For the multiplication procedure (denoted by Mult. of share vectors) with Shamir's scheme, the cost of polynomial evaluations may be lowered by using the discrete Fourier transform as described in [CPR12].

According to this table, our masking procedure is dramatically more efficient that Shamir's secret sharing scheme. When $n$ behaves as a linear function of $t$, our solution is asymptotically the most efficient since the secure multiplication procedure needs a number of field multiplications linear in $t$ while a quadratic number is needed for the Boolean masking scheme. In the next section, we compare these methods for securing AES, with concrete parameters.

| | Our Masking Scheme F2 | Our Masking Scheme F4 |
|---|---|---|
| Add. with a constant | $1$ *add* | |
| Add. of share vectors | $t+1$ *add* | |
| Mult. with a constant | $t+1$ *mult* | |
| Mult. of share vectors | $(n-1)+k(t+1)$ *rand*<br>$[(t+1)(L+n-1)$<br>$+2n-1]$ *add*<br>$n$ *mult* | $(n-1)+k(t+1)$ *rand*<br>$[(t+1)(L+n-1)$<br>$+2n-1]$ *add*<br>$n$ *mult*<br>$(t+1)L'$ *mult. with* $w$ |
| Square of share vectors | $t+1$ *squares* | |

| | Boolean Masking [RP10] | Shamir Masking [GM11,PR11] |
|---|---|---|
| Add. with a constant | $1$ *add* | $t+1$ *add* |
| Add. of share vectors | $t+1$ *add* | |
| Mult. with a constant | $t+1$ *mult* | |
| Mult. of share vectors | $t(t+1)/2$ *rand*<br>$2t(t+1)$ *add*<br>$(t+1)^2$ *mult* | $t(2t+1)$ *rand*<br>$t(2t^2+2t)$ *add*<br>$(2t+1)^2$ *mult*<br>$2t+1$ *polynomial evaluations:*<br>$(2t+1)\times(t^2+t)$*add*<br>$(2t+1)\times(t^2+t)$<br>*mult with const* |
| Square of share vectors | $t+1$ *squares* | |

*Remarks*: $n$ and $k$ corresponds to the parameters of the $[n+1, k+1]$ codes given in Table 1 and 2. L and L' refer respectively to the number of additions and *low-cost multiplication with* $w$ for *encoding* given in Table 1 and 2.

**Table 3.** Complexity of masked operations against $t^{th}$-order SCA

## 5 Application to AES

In this section, we apply our masking scheme to design a secure implementation of AES against $t^{th}$-order SCA and compare its performance with Boolean masking [RP10] and Shamir's secret sharing scheme [GM11,PR11].

The AES [FIP01] is a block cipher algorithm which operates on a $4 \times 4$ bytes state. The bytes are viewed as elements of $\mathbf{F}_{2^8} = \mathbf{F}_2[x]/(x^8+x^4+x^3+x+1)$. During encryption, four transformations are involved. *AddRoundKey* is an addition between the state and the round key, *SubBytes* is a nonlinear transformation, *ShiftRows* and *MixColumns* are linear transformations.

In the following, we describe the implementation of our $t^{th}$-order masking on all the AES transformations.

### 5.1 Secure Implementation of Linear AES Transformations

To secure the linear transformations (*AddRoundKey*, *ShiftRows*, *MixColumns*) against $t^{th}$-order SCA, we propose to apply Boolean masking. More precisely, we consider the $[t+2, t+1]$ parity check code $C^\star$ over $\mathbf{F}_{2^8}$ constructed from the generator matrix given by (2). Each element of the state is shared into $t+1$ elements of $\mathbf{F}_{2^8}$ with the *encoding* procedure described in subsection 3.2. Hence all linear AES transformations can be performed share by share as for Boolean masking. As a result the masking of this transformation is as efficient as the method of [RP10]. More precisely, *AddRoundKey* requires $16 \times (t+1)$ additions in $\mathbf{F}_{2^8}$ and *MixColumns* requires $4 \times 15 \times (t+1)$ additions and $4 \times 4 \times (t+1)$ multiplications by the constant $\{0\text{x}02\}$ which can be performed by a look-up table.

### 5.2 Secure Implementation of *SubBytes* Transformation

The nonlinear *SubBytes* transformation is the composition of two functions: the nonlinear calculation of the inverse in $\mathbf{F}_{2^8}$ and an affine transformation over $\mathbf{F}_2$, denoted $Af$. To secure the computation of the inverse in $\mathbf{F}_{2^8}$, one uses the fact that this operation can be defined as $X \mapsto X^{254}$. As shown in [RP10], this exponentiation requires a lower bound of 4 multiplications and 7 squares over $\mathbf{F}_{2^8}$:

$$X^{254} = [(X^2X)^4(X^2X)]^{16}(X^2X)^4X^2 \ . \tag{3}$$

At the beginning of the *SubBytes* Transformation, we apply the code switching method of subsection 4.3. More precisely, each coordinate of the share vectors of the short code $\mathcal{C}^\star$ are re-encoded in a code $\mathcal{C}$ selected from Table 1 or 2. Then, the entire secure inversion is performed in this code with formula (3).

For the codes of Table 1, the square procedure require $n$ squares performed share by share. Secure multiplications are done with Algorithm 4. For the codes of Table 2, the map $X \mapsto X^4$ can be computed efficiently share by share. The square procedure now needs a re-encoding procedure as described in Remark 2. For this reason, with these codes, we perform the first operation (*i.e.*, the

computation of the share vector of $X^2$) with the short parity check code $C^\star$ before switching code to $\mathcal{C}$.

Finally, we need to apply the affine transformation $Af$ which can be decomposed as $X \mapsto A(X) + b$ where $A$ is linear over $\mathbf{F}_2$ and $b$ is a constant byte. The function $A$ being linear over $\mathbf{F}_2$ (and not $\mathbf{F}_{2^8}$), we propose to compute this step by switching code to the parity check code $\mathcal{C}^\star$. Hence as for Boolean masking (cf [RP10]), this transformation requires roughly only $t+1$ times the cost of a single $A$ evaluation which is performed by using a look-up table.

### 5.3 Implementation Results and Comparisons

In order to give an idea of the global complexity of our masking scheme, we give in Table 4 estimations of the number of cycles needed for different implementations of a $t^{th}$-order masking of the AES Sbox for $t \in \{1, \ldots, 6\}$. These estimations are based on 8051 assembly language with a 8-bit smartcard CPU. In such an environment, generation of a random byte requires 2 cycles, an addition requires 1 cycle, a secure multiplication over $\mathbf{F}_{2^8}$ implemented by using so-called log/alog tables (see for instance [DR02]) requires roughly 20 cycles, and an access to a look-up table (describing the square operations, the multiplication with $w$ and the affine transformation) requires 3 cycles. In particular the magnitude of complexity estimations given in Table 4 has been confirmed by a real implementation of our proposal, with the two first codes of Table 1, to design a first and second-order secure implementation.

| Scheme \ Order $t$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Boolean masking [RP10] | 0.4 | 0.9 | 1.5 | 2.4 | 3.4 | 4.6 |
| Shamir [GM11,PR11] | 1.3 | 4.8 | 11.6 | 22.7 | 39.3 | 62.3 |
| Our masking scheme $\mathbf{F}_2$ | 0.8 | 1.1 | 3.8 | 4.3 | 5.3 | 6.2 |
| Our masking scheme $\mathbf{F}_4$ | 0.5 | - | 2.2 | 2.9 | 5.7 | - |

**Table 4.** Estimation of timing for a secure AES Sbox on an 8-bit smartcard (in thousands of cycles)

From Table 4, we can see that, as $t$ grows, our approach becomes more and more efficient than the method proposed in [GM11,PR11] using Shamir's secret sharing scheme. Furthermore we can remark that the cost of our method is very close to the method using Boolean masking described in [RP10]. However a security flaw in this method has been very recently announced in [CPRR13] where the authors proposed a new solution. For future work, it will be interesting to compare our proposal to this solution and to see if some ideas using the framework of error correcting codes introduced in our work can be adapted to this solution to devise a more efficient masking of AES.

## 6 Conclusion

In this paper, we have presented a new high-order masking scheme and an application to the AES cipher. The masking scheme relies on secret sharing based on carefully chosen non-MDS linear codes and is significantly more efficient than the methods that rely on Shamir's secret sharing scheme. As a result, when applied to the secure implementation of AES, our masking scheme is a more attractive alternative to Boolean masking than Shamir's scheme. Moreover, the comparison given by Table 4 shows that the efficiency of our proposal is very close to Boolean masking and it could open new perspectives in masking scheme design.

## References


BFGV12. Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and Practice of a Leakage Resilient Masking Scheme. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 758–775. Springer, 2012.

BMK04. J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 69–83. Springer, 2004.

BOGW88. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems For Non-Cryptographic Fault-Tolerant Distributed Computation. *Symposium on Theory of Computing*, pages 1–10, 1988.

CC06. Hao Chen and Ronald Cramer. Algebraic Geometric Secret Sharing Schemes and Secure Multi-Party Computations over Small Fields. In S. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *LNCS*, pages 521–536. Springer, 2006.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty Unconditionally Secure Protocols. *Symposium on Theory of Computing*, pages 11–19, 1988.

CCG+07. Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure Computation from Random Error Correcting Codes. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2007*, volume 7237 of *LNCS*, pages 291–310. Springer, 2007.

CDG+08. Ronald Cramer, Vanesa Daza, Ignacio Gracia, Jorge Jiménez Urroz, Gregor Leander, Jaume Martí-Farré, and Carles Padró. On Codes, Matroids, and Secure Multiparty Computation From Linear Secret-Sharing Schemes. *IEEE Transactions on Information Theory*, 54(6):2644–2657, 2008.

CDM00. Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General Secure Multiparty Computation from any Linear Secret-Sharing Scheme. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer, 2000.

CG13. Claude Carlet and Sylvain Guilley. Side-channel indistinguishability. In *HASP '13 Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM New York, 2013.

CGKS12. Claude Carlet, Philippe Gaborit, Jon-Lark Kim, and Patrick Solé. A New Class of Codes for Boolean Masking of Cryptographic Computations. *IEEE Transactions on Information Theory*, 58(9):6000–6011, 2012.

CPR12. J.-S. Coron, E. Prouff, and T. Roche. On the Use of Shamir's Secret Sharing Against Side-Channel Analysis . In Stefan Mangard, editor, *Smart Card Research and Advanced Applications, 11th International Conference – CARDIS 2012*, LNCS. Springer, 2012.

CPRR13. Jean-Sebastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-Order Side Channel Security and Mask Refreshing. In *Fast Software Encryption – FSE 2013*, 2013.

CS90. John H. Conway and Neil J. A. Sloane. A new upper bound on the minimal distance of self-dual codes. *IEEE Transactions on Information Theory*, 36(6):1319–1333, 1990.

DF12. Stefan Dziembowski and Sebastian Faust. Leakage-Resilient Circuits without Computational Assumptions. In R. Cramer, editor, *Theory of Cryptography Conference – TCC 2012*, volume 7194 of *LNCS*. Springer, 2012.

DR02. J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002.

FIP01. FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, November 2001.

FMPR10. Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography – SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, 2010.

GM11. L. Goubin and A. Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In Preneel and Takagi [PT11], pages 79–94.

GO03. Philippe Gaborit and Ayoub Otmani. Experimental Constructions Of Self-Dual Codes. *Finite Fields and Their Applications-Elsevier*, July 2003.

GRR98. R. Gennaro, M. Rabin, and T. Rabin. Simplifed vss and fact-track multiparty computations with applications to threshold cryptography. *Symposium on Principles of Distributed Computing*, pages 101–111, 1998.

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

JPS05. M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 293–308. Springer, 2005.

KHL11. HeeSeok Kim, Seokhie Hong, and Jongin Lim. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In Preneel and Takagi [PT11], pages 95–107.

KJJ99. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

Koc96. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

Mas93. J. Massey. Minimal Codewords and Secret Sharing. In *Sixth Joint Swedish-Russian Workshop on Information Theory*, pages 246–249, 1993.

Mes00. T.S. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.

MS78. F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 1978.

OMPR05. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In H. Handschuh and H. Gilbert, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *LNCS*, pages 413–423. Springer, 2005.

PR10. Emmanuel Prouff and Thomas Roche. Attack on a Higher-Order Masking of the AES Based on Homographic Functions. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology – INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 262–281. Springer, 2010.

PR11. Emmanuel Prouff and Thomas Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In Preneel and Takagi [PT11], pages 63–78.

PT11. Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*. Springer, 2011.

RP10. M. Rivain and E. Prouff. Provably Secure Higher-order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

Sha79. Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

TDG02. Elena Trichina, D. DeSeta, and L. Germani. Simplified Adaptive Multiplicative Masking for AES. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 187–197. Springer, 2002.