Fault-Tolerant Lagrange Representation Multiplication in the Finite Field $GF(2^k)$

Silvana Medo, Serdar Boztaş

***Abstract*— In this paper, we are concerned with protecting finite field computation against active side channel attacks, i.e., fault attacks. We propose a fault tolerant Lagrange representation modular multiplication algorithm for fault tolerant computation in the finite field $GF(2^k)$ for use in public key cryptosystems. Also, we propose a new model for fault attacks to public key cryptosystem. By use of well-known error correcting codes we provide countermeasures to some fault inducing attacks.**

I. Introduction

The arithmetic structure of finite fields is utilized in *public key cryptography* (smart card technology, e-commerce, and internet security), as well as in *coding theory* (error-free communications and data storage). Public key cryptographic applications rely on computation in very large finite fields (with more than 2^{160} elements for Elliptic Curve Cryptography, and more than 2^{1024} elements for RSA). Unfortunately, a single fault in computation can yield an erroneous output, which can then be used by an adversary to break cryptosystem. Since we require high reliability and robustness, fault tolerant finite field computation in the public key cryptosystems is crucial. Security of cryptosystems does not only depend on the mathematical properties; an adversary can attack the implementation rather than algorithmic specification.

[**?**] presents a model for breaking various cryptographic schemes on tamper resistant devices by taking advantage of random hardware faults. [**?**] presents various methods for attacking *public key cryptosystem*, by use of transient faults. [**?**] presents fault-tolerant computation over the integers based on the modulus replication residue number system, which allows modular arithmetic computations over identical channels. [**?**] presents multipliers for fields $GF(2^k)$ whose operations are resistant to errors caused by certain faults. They can correct single errors caused by one, or more faults in the multiplier circuits. [**?**] introduces scaled embedding for Fault-Tolerant Public Key Cryptography based on arithmetic codes and binary cyclic codes in order to achieve robust fault tolerant arithmetic in the finite field.

In this paper we are concerned with protecting finite field computation against active side channel attacks, i.e., fault attacks, where an adversary induces faults into a device, while it executes the correct program. After outlining the fault tolerant multiplication algorithm in $GF(2^k)$, and after proposing a new model for fault attacks to public key cryptosystem, we make use of well-known error correcting codes in order to provide countermeasures to some fault-inducing attacks.

Manuscript received September 25, 2007; revised November 25, 2007.

Silvana Medo is with the School of Mathematical and Geospatial Sciences, RMIT University,Melbourne, Victoria 3000, Australia, e-mail: silvana.medos@ems.rmit.edu.au.

Serdar Boztaş is with the School of Mathematical and Geospatial Sciences, RMIT University,Melbourne, Victoria 3000, Australia, e-mail: serdar.boztas@ems.rmit.edu.au

A. Security of Public Key Cryptosystems

To quantify the security of public key cryptosystems, the concept of *computational security* is widely used. Here, an adversary is assumed to have limited computation time and memory available (polynomial in the input parameters) and the security of the cryptosystem is based on the fact that the problem of breaking the system is reducible to solving a problem that is strongly believed to be computationally hard, such as factoring a product of two large random primes and taking discrete logarithms in a large finite field. As a rule, one assumes that an adversary always has access to all data being transmitted by two communicating parties and exact knowledge of every aspect of the used cryptographic scheme, except for the secret key–this is referred to as the *Kerckhoffs' Principle*. Moreover an adversary can request encryptions of polynomially many (in the size of the input parameters) chosen messages to achieve his objective. This scenario is usually referred to as a *black-box assumption*, since it allows purely theoretical proofs on paper. However, cryptosystems are used in the real world where cryptographic protocols are implemented in software or hardware, obeying laws of physics. The circuits used *leak* information, e.g., power and timing information, over side channels. Thus, one has a *gray box*, where an adversary has access to several side-channels.

II. Multiplication in $GF(2^k)$

[**?**] gives finite field $GF(2^k)$ analogue of the Montgomery multiplication for modular multiplication of integers. Elements of the finite field are considered as a polynomials of degree $< k$, while x^k is used as a *Montgomery factor*. [**?**] extends same idea to the field $GF(p^k)$, $p > 2$, and presents a new multiplication algorithm for the implementation of the elliptic curve cryptography over an optimal extension field $GF(p^k)$, where $p > 2k$. Elements of the field which are in the polynomial representation of degree $\leq k-1$, are represented by their values at sufficiently many points, while $p(x) = \prod_{i=1}^{k}(x-\alpha_i)$ is used as a *Montgomery factor*, where $\alpha_i \in GF(p)$, $p > 2k$ are distinct.

We will show that the Lagrange representation (LR) modular multiplication algorithm [?] is correct for the case when $p = 2$. Our aim is to extend use of this algorithm for fault tolerant

computation in the field $GF(2^k)$ by use of redundancy.
Let the finite binary extension field $GF(2^k)$, be represented as the set of polynomials modulo an irreducible polynomial $f(x)$, $deg(f(x)) = k$, i.e.

$$GF(2)[x]/ < f(x) >= \{a_0 + \ldots + a_{k-1}x^{k-1} | a_i \in GF(2)\} \quad (1)$$

and where $f(\alpha) = 0$, so that

$$GF(2^k) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^k-2}\}. \quad (2)$$

Lemma 1. *LR modular multiplication algorithm (see [?]) can be applied to the field $GF(2^k)$ if and only if finite field is considered as in (**??**) and (**??**).*

Proof: Since inputs to the computation are elements in the LR, and since we require at least $2k$ polynomial evaluations for LR modular multiplication algorithm (see [?]), then by considering elements of the $GF(2^k)$ as in (**??**) and (**??**), each element of $GF(2^k)$ in polynomial representation can be evaluated at least at the $2k$ field elements represented as in (**??**), since $2^k > 2k$ for $k \geq 3$. ■

A. Fault Tolerant Multiplication

Algorithm 1 Fault tolerant LR Modular Multiplication.

Inputs: $a_i, a_i', b_i, b_i' \in GF(2^k)$, poly. eval. at T and T', $f_i \in GF(2^k)$, poly. eval. at T'.
precomputed: $f_i^{'}$ poly. eval. at T, ξ, $c \times c$ matrices ω, ω'.
Output: $r(x) \in GF(2^k)[x]/ < \prod_{i=1}^{c}(x - \alpha_i) >$, $\alpha_i \in T$

1. $(t_1, \ldots, t_k, \ldots, t_c) \leftarrow (a_1, \ldots, a_k, \ldots, a_c) \otimes (b_1, \ldots, b_k, \ldots, b_c)$
2. $(q_1, \ldots, q_k, \ldots, q_c) \leftarrow (t_1, \ldots, t_k, \ldots, t_c) \otimes (f_1^{'}, \ldots, f_k^{'}, \ldots, f_c^{'})$
3. Change of LR: $(q_1, \ldots, q_c) \rightarrow (q_{c+1}, \ldots, q_{2c})$
4. $(r_{c+1}, \ldots, r_{2c}) \leftarrow ((t_{c+1}, \ldots, t_{2c}) \oplus (q_{c+1}, \ldots, q_{2c}) \otimes (f_{c+1}, \ldots, f_{2c})) \otimes (\xi_{c+1}, \ldots, \xi_{2c})$
5. Change of LR: $(r_{c+1}, \ldots, r_{2c}) \rightarrow (r_1, \ldots, r_c)$
6. Lagrange interpolation: $(r_1, \ldots, r_c)$.

To protect multiplication of the finite field we add redundancy by adding more parallel channels then minimum required, i.e. $c > k$, see Fig.1. Thus inputs are evaluated at more distinct points $c > k$ from the set $T = \{\alpha_i \in GF(2^k) \,|\, i \in \{1 \ldots c\}\}$ and $T' = \{\alpha_j \in GF(2^k) \,|\, j \in \{c+1 \ldots 2c\}\}$, $\alpha_i \neq \alpha_j$, where c depends on security required. Inputs are evaluated at additional $c - k$ distinct elements, with the constraint that elements $\alpha_j \in GF(2^k)$ at which polynomials are evaluated are all distinct elements. By adding $c-k$ extra redundant polynomial evaluations computation now happens in the larger direct product ring

$R = GF(2^k)[x]/ < x-\alpha_1 > \times \ldots \times GF(2^k)[x]/ < x-\alpha_c >$, $\alpha_i \in T$, and,

$R' = GF(2^k)[x]/ < x-\alpha_{c+1} > \times \ldots \times GF(2^k)[x]/ < x-\alpha_{2c} >$,

where $\alpha_j \in T'$. Operations used are componentwise multiplication $\otimes$, and componentwise addition $\oplus$. Also,

$$R \cong R[x] = GF(2^k)[x]/ < m(x) >,$$

where $m(x) = \prod_{i=1}^{c}(x - \alpha_i)$, $c > k$, $\alpha_i \in T$.

Inputs $g_i \in GF(2^k)[x]/ < f(x) >$ are evaluated at c distinct elements from the set T and T' i.e.,

$$g_i(x) \rightarrow (g_i(\alpha_1), \ldots, g_i(\alpha_k), \ldots, g_i(\alpha_c)),$$
$$g_i(x) \rightarrow (g_i(\alpha_{c+1}), \ldots, g_i(\alpha_{c+k}), \ldots, g_i(\alpha_{2c})),$$

where $g_i(\alpha_1), \ldots, g_i(\alpha_k)$ and $g_i(\alpha_{c+1}), \ldots, g_i(\alpha_{c+k})$ are *non-redundant components*, and $g_i(\alpha_{k+1}), \ldots, g_i(\alpha_c)$ and $g_i(\alpha_{c+k+1}), \ldots, g_i(\alpha_{2c})$ are *redundant components*.

Firstly, computation is performed in parallel in the field $GF(2^k)$ as in step 1, and 2 of Algorithm **??**. Since, as a *Montgomery factor* we chose polynomial $p(x) = \prod_{i=1}^{c}(x - \alpha_i)$, $\alpha_i \in T$ (see [?]), and since, $r(x) = t(x) + q(x)f^{'}(x)$ is a multiple of $p(x)$, we chose new set of distinct elements $T' = \{\alpha_{c+1}, \ldots, \alpha_{2c}\}$, such that $p'(x) = \prod_{j=c+1}^{2c}(x - \alpha_j)$, $\alpha_j \in T'$ with constraint that $gcd(p(x), p'(x)) = 1$.
Let $q(x) = \sum_{i=1}^{c} q_i \prod_{j=1, i\neq j}^{c} \frac{x-\alpha_j}{\alpha_i-\alpha_j}$, $\alpha_i, \alpha_j \in T$ and if $w_{m,i} = \prod_{j=1,\, j\neq i}^{c} \frac{\alpha_m'-\alpha_j}{\alpha_i-\alpha_j}$, $\alpha_m' \in T'$, $\alpha_i\alpha_j \in T$ then the change of LR from T to T' is achieved by:

$$\begin{pmatrix} q_{c+1} \\ \vdots \\ q_{2c-1} \\ q_{2c} \end{pmatrix} = \begin{pmatrix} w_{1,1} & \ldots & \ldots & w_{1,c} \\ \vdots & \vdots & \vdots & \\ w_{k,1} & \ldots & \ldots & w_{k,c} \\ \vdots & \vdots & \vdots & \\ w_{c,1} & \ldots & \ldots & w_{c,c} \end{pmatrix} \begin{pmatrix} q_1 \\ \vdots \\ q_{c-1} \\ q_c \end{pmatrix}. \quad (3)$$

Therefore, we the have mapping

$$\psi : \phi(R) \rightarrow R'.$$

In step 4, in parallel we compute $r_i = (t_i + q_i f_i)\xi_i$, $i \in \{c+1 \ldots 2c\}$ where $\xi_i = \Pi_{j=1}^{c}(\alpha_i - \alpha_j)^{-1} \, mod \, f(\alpha)$, $\alpha_i \in T'$. Now, to get vector r at T, we do change of LR from T' to T, i.e., let $\omega_{m,i}' = \prod_{j=1,\, j\neq i} \frac{\alpha_m-\alpha_j'}{\alpha_i'-\alpha_j'}$, $\alpha_m \in T$, $\alpha_i', \alpha_j' \in T'$ then

$$\begin{pmatrix} r_1 \\ \vdots \\ r_{c-1} \\ r_c \end{pmatrix} = \begin{pmatrix} w_{1,1}' & \ldots & \ldots & w_{1,c}' \\ \vdots & \vdots & \vdots & \vdots \\ w_{k,1}' & \ldots & \ldots & w_{k,c}' \\ \vdots & \vdots & \vdots & \vdots \\ w_{c,1}' & \ldots & \ldots & w_{c,c}' \end{pmatrix} \begin{pmatrix} r_{c+1} \\ \vdots \\ r_{2c-1} \\ r_{2c} \end{pmatrix}. \quad (4)$$

By Lagrange interpolation (LI), if there are no fault effects, c output components at distinct elements form set T will determine a unique polynomial of degree $\leq k-1$ with coefficients $a_i \in GF(2^k)$, otherwise, it will be of degree $\geq k$.

Definition 2. *The set of correct results of computation is*

$$C = \left\{r(x) \in R[x] \,|deg\,(r(x)) < k,\, a_i \in GF(2^k)\right\}.$$

1) Complexity: Polynomial interpolation is only done at the end of the computation, and its complexity is $O(c^2)$, while complexity of polynomial evaluation is $O(ck)$, $c > k$. Matrices ω, ω' are precomputed, as well as ξ and $f'(x)$.

Lemma 3. *Total computational complexity of the Algorithm* **??** *is* $O(c^2)$, $c > k$.

Proof: In Algorithm **??**, steps 1 and 2 require c operations each, while steps 3 and 4 require $2c^2 - c$ operations each. Also, step 4 require $c^2 + c$. By taking into consideration complexity of polynomial evaluation and interpolation we have that total complexity of the Algorithm **??** is $O(c^2)$. ■

Lemma 4. *Matrices (**??**) and (**??**) satisfy relation* $\omega_{i,j} = \omega'_{i+1,j}$ *iff sets of* c *distinct points are chosen as* $T = \{\alpha^{2m_i} \in GF(2^k) \,|\, i \in \{1, \ldots, c\}\}$, $T' = \left\{\alpha^{2m_i+1} \in GF(2^k) \,|\, i \in \{1, \ldots, c\}\right\}$.

Proof: Let

$$\omega_{i,j} = \prod_{s=1,\, s\neq j}^{c} \frac{\alpha^{2m_i+1} - \alpha^{2m_s}}{\alpha^{2m_j} - \alpha^{2m_s}} \tag{5}$$

and multiply (**??**) by $\alpha^{c-1}/\alpha^{c-1}$ then

$$\begin{aligned} \omega_{i,j} &= \prod_{s=1,\, s\neq j}^{c} \frac{\alpha^{2m_i+2} - \alpha^{2m_s+1}}{\alpha^{2m_j+1} - \alpha^{2m_s+1}} \\ &= \prod_{s=1,\, s\neq j}^{c} \frac{\alpha^{2(m_i+1)} - \alpha^{2m_s+1}}{\alpha^{2m_j+1} - \alpha^{2m_s+1}} \end{aligned}$$

Since $m_{i+1} = m_i + 1$ we have

$$\omega_{i,j} = \prod_{s=1,\, s\neq j}^{c} \frac{\alpha^{2m_{i+1}} - \alpha^{2m_s+1}}{\alpha^{2m_j+1} - \alpha^{2m_s+1}} = \omega'_{i+1,j}. \blacksquare$$

III. Fault Model

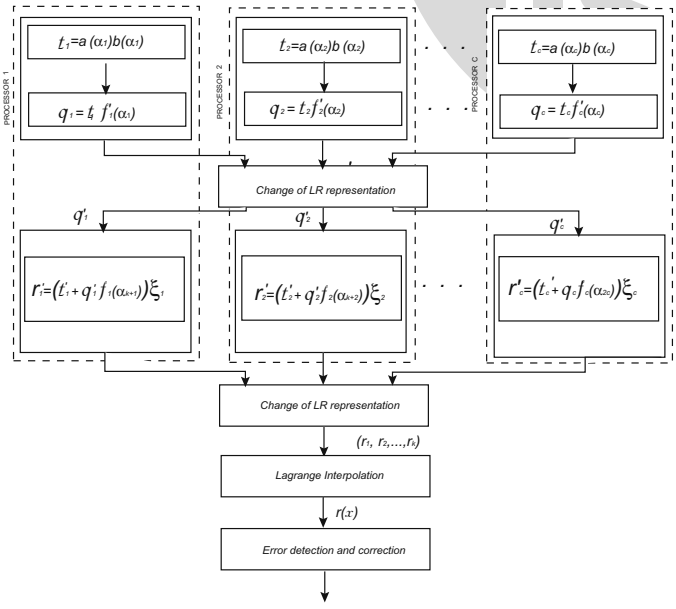

Fig. 1. Fault tolerant computation of the finite field $GF(2^k)$.

There is one processor per independent channel, i.e., see Fig.1. Let us assume that we have c processors, where each processor computes i -th polynomial evaluation and operations of the finite field $GF(2^k)$. Also, we assume that all precomputed inputs are error free, as well as, Lagrange interpolation in step 6.

As a fault attacks we assume methods, approaches and algorithms which when applied to the attacked processor return desired effect. We assume that fault attack induces faults into processors by some physical set up, such that processor is exposed to some physical stress (cosmic rays, heat/infrared radiation, power spikes, clock glitches, ...). An adversary can run the processor several times while inducing faults into structural elements of an attacked processor, till desired effects occur. As a reaction attacked processor malfunctions, i.e., memory cells change their current, bus lines transmit different signals, or structural elements are damaged. It is faulty (does not compute the correct output given its input), and its output is erroneous such that computation assigned to the faulty processor is disturbed, and its channel is affected. We identify memory cells with their values, and we say that faults are induced into variables, or bits. Also, our concern is the effect of a fault as it manifests itself in a modified data, or a modified program execution.
Note that any fault induced in a variable x can be described by means of an additive error term $x \mapsto x' = x + e(x)$ but the error term $e(x)$ can itself take on quite different characteristics, depending on the type of the fault:

Stuck-at Faults Let b be an arbitrary bit stored in memory. Assume that b is modified by a stuck-at fault. Then $b \mapsto b' = c$, where the constant $c = 1$ or $c = 0$. The value of the affected bit is not changed any more, even if a variable x, which uses these bits, is overwritten. Clearly stuck-at faults will have a noticeable effect only if the variable is overwritten at some point.

Bitflip Faults Let b be an arbitrary bit stored in memory. Assume that b is modified by a bitflip fault. Then $b \mapsto b' = b + 1 \pmod 2$. The effect may be transient, permanent or destructive. A bitflip fault is easy to visualize, and always results in a fault on a variable using the bit which is faulty.

Random Faults Let b be an arbitrary bit stored in memory. Assume that b is modified by a random fault. Then $b \mapsto b'$ where b' is a random variable taking on the values 0 or 1. The effect may be transient, permanent or destructive. Since several physical methods of fault induction are difficult to control precisely, random faults are considered to be the most realistic fault type. The random variable which models the fault may be uniform or non-uniform.

Note that the above faults can be considered for an arbitrary but unknown set of bits B, where assumptions about how the adversary controls the choice of B can also model different attack scenarios. Therefore, we assume following fault models (inspired by [**?**]):
Random Fault Model (RFM) - Assume that an adversary does not know much about his induced faults to know its

effect, but he knows the affected variable at specific channel. Therefore, we assume that affected variable $r_j \in GF(2^k)$ at specific channel j is changed to some random value from the finite field $GF(2^k)$, where all values can occur with the same probability.
This model is used if attacker knows that an induced fault will set affected variable to a random value from $GF(2^k)$ according to the uniform distribution, or if his fault attack does not depend on some special values that have to appear at some time. This fault model relays on the *random fault type*.

Arbitrary Fault Model (AFM) - An adversary can target specific line of code at specific channel, but no specific variable in that line, i.e., adversary has limited control over induced faults, does not know much about his induced faults to know its type, or error distribution.
In *AFM*, transient faults on any variable, or operation in the affected line of code at specific channel is the same as if the result of the targeted line of code is changed by some fault at specific channel. In situation of permanent fault, we assume that all variables used in the targeted line of code are hit with the same uniform probability. This attack is successful if attacker does not need the assumptions about distribution of the error value, or does not need to be able to guess the error term to get information. Also, we can assume that an adversary can not target specific line of code, but will hit any line with known probability.

Mathematically, the effect of an attack using these fault models can be modeled as an addition of an unknown error $e_i \in GF(2^k)$. In case of RFM we assume that a variable r_j at specific channel j is changed to some random value $r_j + e_j$, where $e_j \in GF(2^k)$ with the same uniform probability, i.e., fault may result in any faulty value, while for AFM if we let r_i be component to which is assigned the result of the affected line of code, then the faulty value is $r_i + e_i$, where $e_i \in GF(2^k)$, and whose probability distribution is arbitrary and unknown.

Since, computation is decomposed into parallel, mutually independent channels, adversary can use either RFM, or AFM per channel. Assume that at most $c-k$ channels have faults. Let $r' \in R$ be computed vector with c components, where $e_j \in GF(2^k)$ is the error at j -th position; then the computed component at the j-th positions is

$$r_j = r(\alpha_j) + e_j, \tag{6}$$

and each processor will have as an output component

$$r_j = \begin{cases} r(\alpha_j) + e_j, & j \in \{j_1, \ldots, j_t\}, \\ r(\alpha_j), & \text{else.} \end{cases}$$

Here, we have assumed that the set of error positions are $\{j_1, \ldots, j_t\}$, i.e., e_j is the effect of the fault in the channel j_i. By Lagrange interpolation, the computed vector $r' \in R$ with corresponding set of c distinct elements from T gives as a output unique polynomial $r'(x) \in GF(2^k)[x]/ < m(x) >$,

$$\begin{aligned} r'(x) &= \sum_{1\le i\le c} r_i \prod_{1\le j\le c,\, i\neq j} \frac{x-\alpha_j}{\alpha_i-\alpha_j} \\ &= \sum_{1\le i\le c} r_i(\alpha_i) \prod_{1\le j\le c,\, i\neq j} \frac{x-\alpha_j}{\alpha_i-\alpha_j} + \\ &\quad \sum_{1\le l\le t} e_{j_l} \prod_{0\le i\le c-1,\, j_l\neq i} \frac{x-\alpha_i}{\alpha_{j_l}-\alpha_i} \\ &= r(x) + e(x), \end{aligned} \tag{7}$$

where $r(x)$ is correct polynomial of computation of degree $\le k-1$ with coefficients from $GF(2^k)$, and $e(x)$ is the error polynomial such that

Theorem 5. *Let effects of the fault $e_{j_1} \neq 0, \ldots, e_{j_t} \neq 0$ be any set of $1 \le t \le c-k$ elements of $GF(2^k)$, $c > k$, then $deg(e(x)) > k-1$ whose coefficients $a_i \in GF(2^k)$.*

Proof: We have that

$$\begin{aligned} e(x) &= \sum_{1\le l\le t} e_{j_l} \prod_{0\le i\le c-1,\, j_l\neq i} \frac{x-\alpha_i}{\alpha_{j_l}-\alpha_i} \\ &= \prod_{0\le i\le c-1} (x-\alpha_i) \left(\frac{e_{j_1}}{(x-\alpha_{j_1}) \prod_{0\le i\le c-1,\, j_1\neq i} (\alpha_{j_1}-\alpha_i)} \right. \\ &\quad \left. + \ldots + \frac{e_{j_t}}{(x-\alpha_{j_1}) \prod_{0\le i\le c-1,\, j_1\neq i} (\alpha_{j_t}-\alpha_{c-1})} \right). \end{aligned}$$

Since,

$$deg\left(\frac{\prod_{0\le i\le c-1} (x-\alpha_i)}{(x-\alpha_{j_1})} \right) = c-1, \ldots,$$

$$deg\left(\frac{\prod_{0\le i\le c-1} (x-\alpha_i)}{(x-\alpha_{j_t})} \right) = c-1,$$

$c > k$ then $deg\,(e(x)) > k-1$ with coefficients

$$\frac{e_{j_k}}{(x-\alpha_{j_1}) \prod_{0\le i\le c-1\, j_1\neq i} (\alpha_{j_k}-\alpha_i)} \in GF(2^k). \blacksquare$$

Therefore, faulty processors affect the result in an additive manner, see (**??**).

Lemma 6. *The error is masked iff error polynomial $e(x)$ has coefficients from $GF(2^k)$, and if $deg(e(x)) \le k-1$.*

Proof: Let $r'(x)$ be computed polynomial as in (**??**). Since, $deg(r(x)) \le k-1$ with coefficients $a_i \in GF(2^k)$, then if $deg(e(x)) \le k-1$ with coefficients $a_i \in GF(2^k)$ we have that $deg(r'(x)) \le k-1$ with coefficients $a_i \in GF(2^k)$ in which case error is masked. ■

Lemma 7. *Let k be the degree of the finite field, and let $c > k$ be the number of parallel independent channels (or number of processors). Then if up to $c-k$ channels fail, the output polynomial $r'(x)$ is such that $deg(r'(x)) > k-1$ with coefficients $a_i \in GF(2^k)$.*

Proof: By referring to (**??**), since $deg(e(x)) > k-1$ with coefficients $a_i \in GF(2^k)$, and $deg(r(x)) \le k-1$ with coefficients $a_i \in GF(2^k)$, the output polynomial $r'(x)$ has to be such that $deg(r'(x)) > k-1$, and $a_i \in GF(2^k)$. ■

Lemma 8. *Let k be the degree of the finite field, and let $c > k$ be number of parallel independent channels (or number of processors). If there is no faulty processors then there are no errors and $deg(r'(x)) \leq k-1$ and coefficients $a_i \in GF(2^k)$, i.e., $r'(x) = r(x)$.*

Proof: If there are no faulty processors, then clearly no errors occurred, and $deg(r'(x)) \leq k-1$ with coefficients $a_i \in GF(2^k)$, so that $r'(x) = r(x)$. ■

Now, it is straightforward to appeal to standard coding theory results to show that:

Theorem 9. *(i) If the number of parallel, mutually independent, identical redundant channels is $d + t \leq c - k$ $(d \geq t)$, then up to t faulty processors can be corrected, and up to d simultaneously detected. (ii) By adding $2t$ redundant independent channels at most t faulty processors can be corrected.*

While it is true that arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed here, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost. We also remark that the Welch-Berlekamp algorithm is suitable for correcting the faults induced by the attacks described in this paper. Note that to specify the algorithm we choose a set of k indices $K = \{0, 1, \ldots, k-1\}$, and $\overline{K} = \{0, \ldots, c-1\} \setminus K$.

Algorithm 2 Welch-Berlekamp Decoding of the Output Vector.

Inputs: output vector of computation $r' = (r_0, \ldots, r_{k-1}, r_k, \ldots, r_{c-1})$, set of c distinct points $T = \{\alpha_j | \alpha_j \in GF(2^k)\}$, set of indices $K = \{0, 1, \ldots, k-1\}$, polynomial $g(x) = \prod_{i \in K}(x - x_i)$

Outputs: polynomials $d(x)$, $h(x)$.

1. By *Lagrange interpolation*, interpolate output vector r' in order to get polynomial $r'(x)$, if $deg(r'(x)) < k$ and $a_i \in GF(2^k)$ then STOP, else
2. for $i \in K$ find $r'(x)$, where $deg(r') < k$,
3. evaluate $r'(x)$, at α_l, $l \in \overline{K}$,
4. determine syndromes $S_l = r_l - r'(x_l)$, $l \in \overline{K}$,
5. determine $y_l = \frac{S_l}{g(x_l)}$,
6. solve key equation $d(x_l)y_l = h(x_l)$.

Example 10. *Assume that we want to protect computation in the finite binary field $GF(2^5) \cong GF(2)[x]/ < x^5 + x^2 + 1 >$, where α is primitive root of the primitive polynomial $x^5 + x^2 + 1$, i.e., $\alpha^5 + \alpha^2 + 1 = 0$.*
Let the inputs to the computation be the following finite field elements: $a(x) = x^3 + x^2 + 1$, $b(x) = x^3 + 1$. We want to compute following expression $(a(x)b(x)) \bmod f(x)$. Since, $k = 5$, the minimum number of polynomial evaluations is 5, but in order to correct single error, we add $c - k = 2$ extra channels. Therefore, we chose following sets of distinct points

$$T = \{\alpha^{18}, \alpha^{20}, \alpha^{22}, \alpha^{24}, \alpha^{26}, \alpha^{28}, \alpha^{30}\}, \quad \textit{and}$$

$$T' = \{\alpha^{19}, \alpha^{21}, \alpha^{23}, \alpha^{25}, \alpha^{27}, \alpha^{29}, \alpha^{31}\},$$

such that $p(x) = \prod_{i=1}^{7}(x - \alpha_i)$, $\alpha_i \in T$, and $p'(x) = \prod_{i=1}^{7}(x - \alpha_j)$, $\alpha_j \in T'$, and $gcd(p(x), p'(x)) = 1$. At elements of the set T and T' we evaluate inputs, i.e.,

$$a(x) \quad \textit{at } T \textit{ is} \quad a = (\alpha^{27}, \alpha^{30}, \alpha^{8}, \alpha^{18}, \alpha, \alpha^{8}, \alpha^{24}),$$

$$a(x) \quad \textit{at } T' \textit{ is} \quad b' = (\alpha, \alpha^{2}, \alpha^{6}, \alpha^{16}, \alpha^{3}, \alpha^{17}, 1),$$

$$b(x) \quad \textit{at } T \textit{ is} \quad b = (\alpha^{12}, \alpha^{3}, \alpha^{10}, \alpha^{4}, \alpha^{9}, \alpha^{7}, \alpha^{26}),$$

$$b(x) \quad \textit{at } T' \textit{ is} \quad b' = (\alpha^{28}, \alpha^{18}, \alpha^{22}, \alpha^{14}, \alpha^{11}, \alpha^{21}, 0).$$

Also,

$$f(x) \quad \textit{at } T' \textit{ is} \quad f_{T'} = (\alpha^{10}, \alpha^{3}, \alpha^{27}, \alpha^{5}, \alpha^{29}, \alpha^{30}, 1).$$

$$f'(x) \quad \textit{at } T \textit{ is} \quad f'_{T} = (\alpha, \alpha^{8}, \alpha^{19}, \alpha^{6}, \alpha^{14}, \alpha^{13}, \alpha^{16}),$$

$$\xi = (\alpha^{23}, \alpha^{2}, \alpha^{7}, 1, \alpha^{12}, \alpha^{19}, \alpha^{22}).$$

Interpolation matrices are

$$\omega = \begin{pmatrix} \alpha^{23} & \alpha^{30} & \alpha^{5} & \alpha^{19} & \alpha^{2} & \alpha^{12} & \alpha^{22} \\ \alpha^{2} & \alpha^{19} & \alpha^{4} & \alpha^{11} & \alpha^{10} & \alpha^{25} & \alpha^{13} \\ \alpha^{24} & \alpha^{3} & \alpha^{29} & \alpha^{15} & \alpha^{7} & \alpha^{7} & 1 \\ \alpha^{11} & \alpha^{6} & \alpha^{25} & \alpha^{21} & \alpha^{23} & \alpha^{16} & \alpha^{25} \\ \alpha^{5} & \alpha^{5} & \alpha^{9} & \alpha^{29} & \alpha^{10} & \alpha^{13} & \alpha^{15} \\ \alpha^{26} & \alpha^{4} & \alpha^{13} & \alpha^{18} & \alpha^{23} & \alpha^{5} & \alpha^{17} \\ \alpha^{28} & \alpha^{29} & \alpha^{16} & \alpha^{26} & \alpha^{16} & \alpha^{22} & \alpha^{13} \end{pmatrix},$$

$$\omega' = \begin{pmatrix} \alpha^{7} & \alpha^{4} & \alpha^{17} & \alpha^{15} & \alpha^{24} & \alpha^{25} & \alpha^{12} \\ \alpha^{23} & \alpha^{30} & \alpha^{5} & \alpha^{19} & \alpha^{2} & \alpha^{12} & \alpha^{22} \\ \alpha^{2} & \alpha^{19} & \alpha^{4} & \alpha^{11} & \alpha^{10} & \alpha^{25} & \alpha^{13} \\ \alpha^{24} & \alpha^{3} & \alpha^{29} & \alpha^{15} & \alpha^{7} & \alpha^{7} & 1 \\ \alpha^{11} & \alpha^{6} & \alpha^{25} & \alpha^{21} & \alpha^{23} & \alpha^{16} & \alpha^{25} \\ \alpha^{5} & \alpha^{5} & \alpha^{9} & \alpha^{29} & \alpha^{10} & \alpha^{13} & \alpha^{15} \\ \alpha^{26} & \alpha^{4} & \alpha^{13} & \alpha^{18} & \alpha^{23} & \alpha^{5} & \alpha^{17} \end{pmatrix}.$$

Therefore,

$$t_T = (\alpha^{8}, \alpha^{2}, \alpha^{18}, \alpha^{22}, \alpha^{10}, \alpha^{15}, \alpha^{19}) \quad \textit{and}$$

$$q_T = (\alpha^{9}, \alpha^{10}, \alpha^{6}, \alpha^{28}, \alpha^{24}, \alpha^{28}, \alpha^{4}).$$

By change of LR from $T \to T'$ we have

$$q'_{T'} = (\alpha^{24}, \alpha^{13}, \alpha^{11}, \alpha^{16}, \alpha^{22}, \alpha^{17}, \alpha^{30}), \quad \textit{such that}$$

$$r'_{T'} = (\alpha^{23}, \alpha^{28}, \alpha^{8}, \alpha^{6}, \alpha^{22}, \alpha^{11}, \alpha^{21}).$$

By change of LR from $T' \to T$ we have

$$r_T = (\alpha^{23}, \alpha^{12}, \alpha^{4}, \alpha^{9}, \alpha^{19}, \alpha^{3}, \alpha^{22}). \tag{8}$$

*By interpolating (**??**) at distinct points of T we have*

$$r(x) = \alpha^{30}x^4 + \alpha^{26}x^3 + \alpha^{28}x^2 + \alpha^{24}x + \alpha^{22}.$$

Assume that an adversary induces faults (either RFM, or AFM) into one of 7 processors, by some pysical set up, causing attacked processor to be faulty, such that erroneous output of the computation is, i.e. $r = (\alpha^{23}, \alpha^{12}, \underline{\alpha^{2}}, \alpha^{9}, \alpha^{19}, \alpha^{3}, \alpha^{22})$. Now, we select set of $k = 5$ indices $K = \{0, 1, 2, 3, 4\}$ such that by interpolating

$$r = (\alpha^{23}, \alpha^{12}, \alpha^{2}, \alpha^{9}, \alpha^{19}) \quad \textit{at} \quad (\alpha^{18}, \alpha^{20}, \alpha^{22}, \alpha^{24}, \alpha^{26})$$

we get

$$r\,(x) = \alpha^{7}x^4 + \alpha^{18}x^3 + x + \alpha^{23}x^2 + \alpha^{30}x + \alpha^{3}.$$

Given index selection K we evaluate $r(x)$ at $\alpha_5 = \alpha^{28}$, $\alpha_6 = \alpha^{30}$, i.e., $r(\alpha^{28}) = \alpha^{14}$, $r(\alpha^{30}) = \alpha^{12}$, such that $S = \left(0,0,0,0,0,\alpha^{22},\alpha^{16}\right)$. Therefore, syndromes are $S_5 = \alpha^{22}$, $S_6 = \alpha^{16}$. Let now define polynomial

$$g(x) = \left(x-\alpha^{18}\right)\left(x-\alpha^{20}\right)\left(x-\alpha^{22}\right)\left(x-\alpha^{24}\right)\left(x-\alpha^{26}\right).$$

New interpolated data is given by $\alpha_5 = \alpha^{28}$, $\alpha_6 = \alpha^{30}$, and

$$y_5 = \frac{S_5}{g(\alpha_5)} = \alpha, \quad \text{and} \quad y_6 = \frac{S_6}{g(\alpha_6)} = \alpha^8.$$

The problem is to determine polynomials $d(x), h(x)$ from $d(\alpha_5)\, y_5 = h(x_5)$, $d(\alpha_6)\, y_6 = h(\alpha_6)$. By rational interpolation at points $\left(\alpha^{28},\alpha\right), \left(\alpha^{30},\alpha^8\right)$ we obtain that $d(x) = \alpha^{26} + x\alpha^4$ and $h(x) = \alpha^{23}$.
Therefore, error locations are the roots of the polynomial $d(x)$, i.e., $\alpha_2 = \alpha^{22}$, while error values are obtained by

$$S(x) = \frac{h(x)\, g(x)}{d(x)} = \alpha^{19}x^4 + \alpha^7 x^3 + \alpha^{25}x^2 + \alpha^{20}x + \alpha^{14},$$

such that $e_2 = S_2 - S\left(\alpha^{22}\right) = \alpha^7$, so that $r_2 - e_2 = \alpha^2 - \alpha^7 = \alpha^4$. Therefore, the correct output vector of computation is

$$r = \left(\alpha^{23},\alpha^{12},\alpha^4,\alpha^9,\alpha^{19},\alpha^3,\alpha^{22}\right). \blacksquare$$

IV. Conclusions

We have described fault attacks on cryptosystems and proposed a means of protecting computation of the finite field $GF(2^k)$ against side-channel attacks, by decomposing computation over parallel, independent, identical channels. This offers a great advantage, since computations are mutually independent (fault effects do not spread to the other channels), and they are performed over the same field. Fault-tolerant computation is obtained by the use of redundancy. By adding $d+t$, $d \geq t$ redundant channels we can correct up to t faulty processors, and simultaneously detect d faulty processors. Either of two proposed fault models, *RFM*, or *AFM* can be used on each channel. Our method covers random and burst errors that can be caused by malicious fault insertion by an adversary, or transient faults. Also, efficient error correction is possible through the use of *Welch-Berlekamp decoding algorithm*. Moreover, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost.

In current work, we are directly applying the method developed in this paper to the algorithm specific computations which are used in elliptic and hyperelliptic curve cryptosystems. Since the group addition in such cryptosystems is built up of a specific sequence of finite field additions and multiplications–to which the results of this paper directly apply–this is a natural progression in our research.

Acknowledgment

The first author would like to thank the Australian Research Council for its support through the ARC Linkage grant, LP0455324.

References

[1] R. Anderson, M. Khun, "Tamper Resistance - a Cautionary Note", *Proceedings of the Second Usenix Workshop on Electronic Commerce*, Vol. 2, pp.1-11, Novembar 1996.
[2] J.C.B. Bajard, L.Imbert, C.Negre and T. Plantard, " Efficient Multiplication $GF(p^k)$ for Elliptic Curve Cryptography", *Proceedings of the 16th IEEE Symposium on Computer Arithmetic(ARITH'03)*, pp. 182, 2003.
[3] F. Bao, R. H. Deng, Y. Han, A. B. Jeng, A.D. Narasimhalu and T-H. Ngair, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults", Security Protocols, *Lecture Notes in Computer Science*, Vol. 1361, pp.115-124, Springer Verlag, 1997.
[4] P. E. Beckmann and B. R. Musicus, "Fast Fault-Tolerant Digital Convolution Using a Polynomial Residue Number System", *IEEE Transactions on Signal Processing*, Vol. 41, No. 7, pp. 2300–2313, 1993.
[5] D. Boneh, R. A. DeMilo, R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations", *Journal of Cryptology*, Vol. 14, pp.101-119, 2001.
[6] J. Gathen and J. Gerhard, " Modern Computer Algebra", *Cambridge University Press*, UK, 1999.
[7] G. Gaubatz and B. Sunar, "Robust Finite Field Arithmetic for Fault-Tolerant Public-Key Cryptography", *Workshop on Fault Diagnosis and Tolerance in Cryptography*, Edinburgh, Scotland, September 2005.
[8] L. Imbert, L. S. Dimitrov, and G. A.Jullien, " Fault-Tolerant Computation Over Replicated Finite Rings", *IEEE Transaction on the Circuits Systems-I: Fundamental Theory and Applications*, Vol. 50, No. 7, July 2003.
[9] P. C. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis", CRYPTO'99, *Lecture Notes in Computer Science*, Vol. 1966, pp.388-397, Springer Verlag, 1999.
[10] Ç. K. Koş and T.Acar, " Montgomery Multiplication in $GF(2^k)$", *Design, Codes and Cryptography*, Vol. 14(1), pp.57–69, April 1998.
[11] R. Lidl and H. Niederreiter, " Introduction to Finite Fields and Their Applications", *Cambridge University Press*, London, 1986.
[12] M. Otto, "Fault Attacks and Countermeasures", PhD Thesis, December, 2004.
[13] I. S. Reed and G. Solomon, " Polynomial Codes over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 8, No. 2, pp. 300–304, June 1960.
[14] L. Welch and E. R. Berlekamp, "Error corrections for algebraic block codes", *U.S. Patent 4 633 470*, September 1983.
[15] S. B. Wicker and V. K. Bhargava, "Reed-Solomon Codes and Their Applications", *IEEE Press*, New York, 1994. Codes", *IEEE VTC 54-th*, Vol. 3, pp. 1472-1476, 2001.
[16] A. Reyhani-Masoleh, M. A. Hasan, "Towards Fault-Tolerant Cryptographic Computations over Finite Field", *ACM Transaction on Embedded Computing Systems*, Vol. 3, No. 3, pp.593-613, August 2004.