# Robust Residue Codes for Fault-Tolerant Public-Key Arithmetic

G. Gaubatz, B. Sunar and M. G. Karpovsky
{gaubatz,sunar}@wpi.edu
{markkar}@bu.edu

**Abstract**

We present a scheme for robust multi-precision arithmetic over the positive integers, protected by a novel family of non-linear arithmetic residue codes. These codes have a very high probability of detecting arbitrary errors of any weight. Our scheme lends itself well for straightforward implementation of standard modular multiplication techniques, i.e. Montgomery or Barrett Multiplication, secure against active fault injection attacks. Due to the non-linearity of the code the probability of successfully injecting an error does not depend on the error pattern itself, but also on the data, which is not known to the adversary a priori. We give a proof of the robustness of these codes by providing an upper bound on the number of undetectable errors.

## 1 Introduction

In 1996 Boneh et al. demonstrated painfully how vulnerable straightforward implementations of public-key cryptographic algorithms are to a class of attacks now commonly referred to as "Bellcore attacks". In the following years several simple and "low-cost" countermeasures were proposed by various authors, most of which were shown to be flawed, depending on the concrete assumptions. Several more involved protection schemes were proposed and broken, which demonstrates the need for a robust error detection scheme. A family of systematic non-linear error detecting codes derived from systematic linear codes [3] has been investigated for use in symmetric ciphers like the AES, e.g. in [4] and later refined in [6]. While these codes work well in the context of symmetric-key algorithms that employ only little more than table look-ups, XORs and byte-wise rotations, they are virtually unusable within the finite field arithmetic structure that forms the basis of most public-key algorithms.

During the early years of fault-tolerant computing, residue codes were proposed [9] as a means for checking arithmetic operations for errors, while preserving the arithmetic structure between operands and their check symbols. The check symbol in residue codes is computed as the remainder of the operand (or its negative) with respect to the check modulus, usually a prime. Several variations such as multi-residue and non-separate (AN) codes were also introduced early on. Designed for the purpose of detecting only sporadically occuring bit errors their arithmetic distance is limited to 2 or 3. Mandelbaum [7] introduced arithmetic codes with larger distance properties, however, with unattractively large redundancy. Unfortunately, due to the linearity of encoding arithmetic codes do not offer robustness properties, since any error pattern which itself is a codeword can not be detected, irrespective of the actual data.

In this paper we propose a non-linear variant of systematic arithmetic residue codes. Our codes are attractive due to their data dependent and asymptotically low probability of missing errors. These properties make it nearly impossible for an adversary to successfully inject faults that are missed by the error detection network.

## 2 Adversarial Fault Model

An active side channel attack such as differential fault analysis (DFA) relies on the manifestation of injected faults as erroneous results which can then be observed at the output of the device. The error is therefore the difference between the expected output $x$ and the observed output $\tilde{x} = x + e$. In the following we do not assume that the adversary is limited to any specific method of fault injection. The only assumption is that direct invasive access to the chip itself is prevented by some tamper-proof coating, a reasonable assumption, since this is a common practice, e.g. in the smart card industry.

However, even if the attacker should manage to remove the shielding and obtain direct access to the chip's surface [1], a successful fault analysis is still highly unlikely. Let us assume for a moment that he has the ability to toggle the state of an arbitrary number of bits with the required spatial and temporal resolution, i.e. reliably introduce an arbitrary error vector. Due to the data dependent probability $Q(e)$ of missing an error in the error detection network, the expected number of attempts to successfully introduce a non-detectable error is at least $\frac{1}{2}\max_{e\neq 0}(Q(e))$. For a sufficiently large digit size $k$ (e.g. 32 bits), this number is on the order of several hundred million trials. While this number seems low enough to warrant an exhaustive trial and error process, such an attack can easily be defeated by a mechanism that detects an unusually large number of errors and simply shuts down the device. Only if the attacker has the capacity to read out the live state of the circuit and instantly compute an undetectable error vector the attack will be successful. We note that these are rather strong assumptions that require a high degree of sophistication and motivation.

When talking about errors as manifestations of faults, there are two principle ways of characterization. A logical error is a bitwise distortion of the data, usually modeled as the XOR of data and error, i.e. $\tilde{x} = x \oplus e$, while arithmetical errors admit the propagation of carries up to the range limit: $\tilde{x} = x + e \bmod 2^k$, where $k$ is the width of the data path. The former is appropriate for storage dominated devices (register files, RAM, flip-flops, etc.), the arithmetic error model is more useful for arithmetic circuits such as adders and multipliers. For the remainder of this paper we will assume the latter, since it helps to simplify the analysis.

## 3 Robust Arithmetic Codes

As mentioned before, a class of non-linear systematic error detecting codes, so-called "robust codes", were proposed by Karpovsky and Taubin [3]. They achieve optimality according to the minimax criterion, that is, they minimize over all $(n, k)$ codes the maxima of the fraction of undetectable errors $Q(e)$ for $e \neq 0$. While they are suitable for data transmission in channels with unknown characteristics, and also for robust implementation of symmetric-key cryptosystems with little arithmetic structure, they do not preserve arithmetic. We thus propose a new type of non-linear arithmetic code, based on the concept of arithmetic residue codes. We define robustness as follows:

**Definition 1.** *Let $C = \{(x, w) | x \in \mathbb{Z}_{2^k}, w = f(x) \in \mathbb{F}_p\}$ be an arithmetic single-residue code with a function $f : \mathbb{Z}_{2^k} \mapsto \mathbb{F}_p$ to compute the check symbol $w$ with respect to the prime check modulus $p$ of length $r = \lceil \log_2 p \rceil$ bits. A non-zero error $e \in \{(e_x, e_w) | e_x \in \mathbb{Z}_{2^k}, e_w \in \mathbb{Z}_{2^r}\}$ is masked for a message $x$, when $(x + e_x, w + e_w) \in C$, i.e. iff*

$$f\left((x + e_x \bmod 2^k)\right) = f(x) + e_w \bmod 2^r \ . \tag{1}$$

*The error masking probability for a given non-zero error is thus*

$$Q(e) = \frac{|\{x | (x + e_x, w + e_w) \in C\}|}{|C|} \ . \tag{2}$$

*We call the code $C$* robust, *if it minimizes maxima of $Q(e)$ over all non-zero errors. Total robustness is achieved for $\max_{e \neq 0}(Q(e)) = 2^{-r}$. We also call $C$ $\epsilon$-robust if it achieves an upper bound $\max_{e \neq 0}(Q(e)) \leq \epsilon \cdot 2^{-r}$, where $\epsilon$ is a constant much smaller than $2^r$.*

In the following we propose a class of non-linear single-residue arithmetic codes $C_p$ based on a quadratic residue check symbol, which achieves $\epsilon$-robustness. Since in practice total robustness is hard to achieve, we will from now on refer to $\epsilon$-robustness simply as robustness.

**Theorem 1** (Robust Quadratic Codes)**.** *Let $C_p$ according to Definition 1, with $f(x) := x^2 \bmod p$. $C_p$ is robust iff $r = k$ and $2^k - p < \epsilon$, and has the error masking equation*

$$(x + e_x \bmod 2^k)^2 \bmod p = w + e_w \bmod 2^k \tag{3}$$

*Proof.* To prove robustness we proceed by proving an upper bound $\epsilon$ on the number of solutions of the error masking equation (3), as that directly translates into a bound on $Q(e)$. The modulo $2^k$ operator from the LHS of (3) stems from the limitation of the data path to $k$-bits. This limits the ranges of both the message and the message error to $0 \leq x, e_x < 2^k$. We can therefore remove the modulo $2^k$ operator by distinguishing between the two cases $x + e_x < 2^k$ and $x + e_x \geq 2^k$. Similarly, an error is masked only if the faulty check symbol $w < p$, so for $k = r$ we can distinguish between the three cases $w + e_w < p$, $p \leq w + e_w < 2^k$ and $2^k \leq w + e_w < 2^k + p$. This allows us to simplify the RHS of (3).

1. Solutions $x < 2^k - e_x$: An error $(e_x, e_w)$ is masked iff

$$(x + e_x)^2 \bmod p = w + e_w \bmod 2^k$$

   Simplifying the RHS we have the following three cases:

   (a) If $w < p - e_w$, the error is masked iff

$$(x + e_x)^2 \bmod p = w + e_w \tag{4}$$

   If $e = (p, 0)$ eq. (4) has exactly $2^k - p$ solutions. For $e_x \neq p$ and $e_x \geq 2^k - p$ there exists at most a single solution; at most two solutions exist in the case of $e_x < 2^k - p$.

   (b) If $p - e_w \leq w < 2^k$, the error will never be masked, since a check symbol $w \geq p$ will always be detected.

   (c) For $w \geq 2^k - e_w$ the error will be masked iff

$$(x + e_x)^2 \bmod p = w + e_w - 2^k \ . \tag{5}$$

   Eq. (5) has at most two solutions.

2. Solutions $x \geq 2^k - e_x$: An error $(e_x, e_w)$ is masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w \bmod 2^k$$

   For the RHS we distinguish the following three cases:

   (a) If $w < p - e_w$, the error is masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w \tag{6}$$

   Eq. (6) has at most two solutions, unless we have an error $e = (2^k - p, 0)$, in which case there are $2^k - p$ solutions.

   (b) If $p - e_w \leq w < 2^k$, the error will never be masked, since a check symbol $w \geq p$ will always be detected.

   (c) For $w \geq 2^k - e_w$ the error will be masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w - 2^k \ . \tag{7}$$

   Eq. (7) has at most two solutions.

$Q(e)$ is determined by the number of solutions to the error masking equation (3). A simple counting argument involving the cases above provides us with an initial, but somewhat weak bound:

> There are at most $2^k - p + 2$ solutions to (3) for errors of the form $(p, 0)$ or $(2^k - p, 0)$, and at most 8 solutions for all other errors.

A tighter bound can be established by differentiating more precisely in which cases two solutions can occur. We omit the proof here due to space restrictions and only give the result.

> There are at most $2^k - p + 1$ solutions for errors of the form $e = (p, 0)$ or $e = (2^k - p, 0)$, and 4 solutions for all other error patterns.

We thus have $\max_{e \neq 0}(Q(e)) = 2^{-k} \cdot \max(4, 2^k - p + 1)$ $\square$

The existence of practical robust codes for cryptographic purposes can be illustrated with the help of the prime number theorem. The idea here is that for fault-tolerance in an adversarial situation, the probability of not detecting an error should be insignificantly small. As we saw from the proof, in the best case we have a probability of at most $Q(e) = 4 \cdot 2^{-k} = 2^{-k+2}$ of not detecting an error (assuming a uniform distribution of messages). Therefore, a $Q(e)$ that makes insertion of an error infeasible for an attacker, requires a sufficiently large digit size $k$ and a prime $p$ close enough to $2^k$ so that the difference does not increase $Q(e)$ too much. For example, for $k = r = 32$ the $k$-bit prime closest to $2^k$ is $2^{32} - 5$, thus bounding $Q(e)$ by $(2^k - p + 1) \cdot 2^{-k} = 3 \cdot 2^{-31}$. Asymptotically, the prime number theorem guarantees the existence of a prime $p$ close enough to $2^k$.

## 4 Robust Arithmetic Operations

In the previous section we proved the robustness of quadratic codes for digits of size $k$ bits. We now wish to apply them in a generalized framework for multi-precision arithmetic over the positive integers.

Due to the range limitation of the information bits to $0 \leq x < 2^k$, we need to handle any overflow resulting from arithmetic operations. This may be a carry bit generated by the addition of two $k$-bit operands, or the $2k$-bit result of a multiplication. The new digits that are created in this manner will need their own check symbols, which cannot be derived from the input operands' check symbols alone. Thus they need to be derived purely from the information bits of the new digits, creating a potential loophole for the insertion of an error. This can be avoided by re-computing the joint check symbol from the newly generated individual check symbols and comparing it to the output of the predictor. This re-computation represents an integrity check which allows us bridge discontinuities introduced by interleaving mixed modulus operations, here the check modulus $p$ and the implicit range limiting modulus $2^k$. Once the integrity check is in place we can perform standard arithmetic operations and implementing an algorithm like Montgomery's for modular arithmetic is straightforward.

In the following we show how this check may be implemented for various arithmetic primitives. Let $(a, |a^2|_p)$ and $(b, |b^2|_p)$ denote encoded input operands $a$ and $b$, where $|x^2|_p$ is short-hand notation for $x^2 \bmod p$. We also introduce mnemonics for these primitives, in order to tie them into a robust variant of the digit serial Montgomery multiplication algorithm in the next section.

**Addition (RADD and RADDC):** RADD (Robust ADDition) and RADDC (Robust ADDition with Carry) compute the sum of the two input operands. This is depicted in Figure 1a. For reference, the operators $\oplus_p$ and $\otimes_p$ stand for addition and multiplication modulo $p$, respectively. The sum $c = a + b$ $(+c_{\text{in}})$ may be larger than $2^k$ by at most a single bit. Let $c_h$ denote this new carry, and $c_l$ the $k$-bit sum. The predictor computes the joint check symbol $|c^2|_p$ as the sum of the check symbols and the product of the operands $|c^2|_p = |(a+b+c_{\text{in}})^2|_p = ||a^2|_p + |b^2|_p + 2(ab + c_{\text{in}}(a+b)) + c_{\text{in}}|_p$. For error detection we first create the check symbol for the $k$-bit sum $|c_l^2|_p$ (the check symbol for the carry bit is the carry bit itself). Then we re-compute the joint check symbol as

$$\begin{aligned} |c^2|_p^* &= |(c_h 2^k + c_l)^2|_p \\ &= \big|c_h \cdot |2^{2k}|_p + c_h \cdot |c_l|_p \cdot |2^{k+1}|_p + |c_l^2|_p\big|_p && (8) \\ &= \big|c_h \cdot |2^{2k} + c_l \cdot 2^{k+1}|_p + |c_l^2|_p\big|_p && (9) \end{aligned}$$

If the check $|c^2|_p^* = |c^2|_p$ holds, then the result is deemed to be free from errors. The resulting carry from both RADD and RADDC is held in a register local to the addition circuit. If the following addition operation is RADDC, then that carry is used for computation of the new sum. If it is RADD, then a zero carry is used.

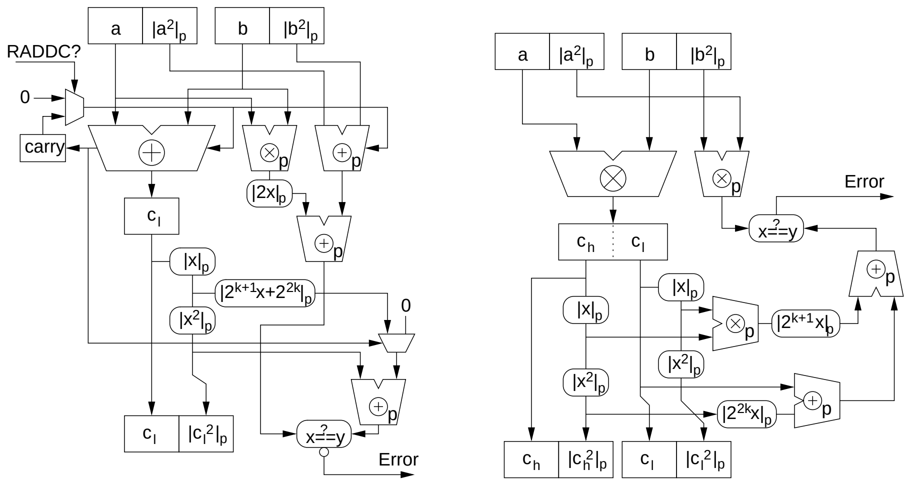
**Multiplication (RMUL):** The product of $a$ and $b$ and its joint check symbol is $(c, |c^2|_p) = (a \cdot b, ||a^2|_p \cdot |b^2|_p|_p)$. However, the previous tuple is not a code word, since $c$ may exceed $2^k$. We therefore split $c$ into two halves $c_h$ and $c_l$, both of which are within the desired range:

$$c = c_h \cdot 2^k + c_l \quad 0 \le c_h, c_l < 2^k \ .$$

We then compute the check symbols $|c_h^2|_p$ and $|c_l^2|_p$ separately, and establish their integrity with the composite check symbol $|c^2|_p$:

$$\begin{aligned} |c^2|_p^* &= \big|(c_h \cdot 2^k + c_l)^2\big|_p \\ &= \big|c_h^2 \cdot 2^{2k} + c_h \cdot c_l \cdot 2^{k+1} + c_l^2\big|_p \\ &= \big||c_h^2|_p \cdot |2^{2k}|_p + |c_h|_p \cdot |c_l|_p \cdot |2^{k+1}|_p + |c_l^2|_p\big|_p \end{aligned}$$

Observe that the values $|2^{2k}|_p$ and $|2^{k+1}|_p$ are constant for a given implementation and that $|c_h|_p$ and $|c_l|_p$ are intermediate results from the computation of



(a) (b)

Figure 1: Robust addition (a) and multiplication (b)

the separate halves' check symbols. Hence we have all the necessary ingredients to re-compute the joint check symbol as $|c^2|_p^*$ and compare it to the instance obtained from the predictor. If the comparison passes we assume there were no errors.

**Shifts, Subtraction, Logic Operations:** We can apply similar re-computation techniques for other operations. Out of space considerations and since we do not need these other operations for the next section, we skip their details at this point.

**Error Detection:** The comparison between the predictor output and the re-computed joint check symbol is an easy target for an attack if carelessly implemented. We therefore require implementation as a totally self-checking checker [8]. The same holds for any other integrity checks.

## 5 Robust Montgomery Multiplication

We now show how to apply our code to a digit serial Montgomery Multiplication scheme. A good overview over several variants of the Montgomery algorithm is given in [5]. In this example we will refer to the finely integrated operand scanning (FIOS) variant. It is the most suitable one for hardware implementations since it can be used in a pipelined fashion offering some degree of parallelization [2].

In the following we require some basic familiarity on part of the reader with the way of how Montgomery multiplication works. To review briefly: the objective is to compute the modular product of two $N$-bit numbers with respect to the $N$-bit modulus $M$. Montgomery's algorithm requires the initial transformation of all operands into residues of the form $\hat{x} = xR \bmod M$ and some final transformation back $x = \hat{x}R^{-1} \bmod M$. Here $R$ is the Montgomery radix, usually $2^{k \cdot e}$, where $e = \lceil N/k \rceil$ represents the number of digits per operand. Without loss of generality we assume that the transformation into the Montgomery residue system has already taken place and we operate entirely within the residue system, so in order to simplify notation we will refer to a residue $\hat{x}$ simply as $x$.

The $k$-bit digit serial FIOS Montgomery algorithm (Alg. 1) takes as its inputs the $e$-digit vectors $a$ and $b$, and computes the product $\mathrm{MM}(a, b) = a \cdot b \cdot R^{-1}$ mod $M$. The value $M'_0$ is pre-computed whenever the modulus changes. In terms of notation, a pair $(C, S)$ represents the concatenation of two variables as the destination for the result of an operation. Furthermore, the variable $C$ is slightly larger than the other variables, i.e. $k + 1$ bits. This is so to efficiently handle extra carries from the accumulation of $C$, $d_i$ and the two products $a_i b_j$ and $M_i U$. The division by $R$ is handled implicitly by the algorithm. For example, in line 4 the sum $(C, S) + M_0 U$ is assigned to $(C, S)$, but in the following step $S$ is dropped. This shift to the right by $k$ bits, repeated $e$ times, results in division by $R$. As one can easily verify, Algorithm 1 consists only of very basic

**Algorithm 1** $k$-bit Digit-Serial FIOS Montgomery Multiplication

```
Require: $d = \{0, \ldots, 0\}$, $M'_0 = -M_0^{-1} \bmod 2^k$
1: for $j = 0$ to $e - 1$ do
2:     $(C, S) \Leftarrow a_0 b_j + d_0$
3:     $U \Leftarrow S M'_0 \bmod 2^k$
4:     $(C, S) \Leftarrow (C, S) + M_0 U$
5:     for $i = 1$ to $e - 1$ do
6:         $(C, d_{i-1}) \Leftarrow C + a_i b_j + M_i U + d_i$
7:     end for
8:     $(d_e, d_{e-1}) \Leftarrow C$
9: end for
```

addition and multiplication steps. We may therefore obtain a robust digit-serial Montgomery algorithm (Alg. 2) simply by mapping all arithmetic steps to our robust arithmetic primitives introduced in the previous section. Additionally we insert intermediate checks during which we verify the integrity of operand values and their check symbols. This is indicated by a call to the pseudofunction $\mathrm{Check}((x, |x^2|_p), \ldots)$. Although not indicated in the algorithm description, we assume further that the error signal generated by the the internal integrity check within the arithmetic primitives RADD, RADDC and RMUL, is also constantly evaluated. In the case of an error the algorithm is aborted by an error handler.

Some comments about the robust algorithm: Algorithm 2 appears much longer than Algorithm 1, since the latter combines multiple arithmetic operations into a single step. Also, while algorithm 1 handles carries implicitly using a larger variable $C$, the robust algorithm is restricted to a digit size of exactly $k$ bits. Thus, extra carry handling steps are required. In Alg. 2, line 6, the destination of the top half of the result is not assigned: $(-,-)$. This is equivalent to computing the result modulo $2^k$, as in Alg. 1, line 3. Similarly in Alg. 2, line 8, where the lower half of the result is dropped due to the implicit shift to the right. The point of performing the addition is purely to determine whether or not a carry is generated.

**Algorithm 2** Robust Montgomery Multiplication

```
Require: $d = \{(0,0),\dots,(0,0)\}$, $M_0' = -M_0^{-1} \bmod 2^k$
 1: for $j = 0$ to $e-1$ do
 2:   if Check($(a_0,|a_0^2|_p),(b_j,|b_j^2|_p),(d_0,|d_0^2|_p),(M_0',|(M_0')^2|_p),(M_0,|M_0^2|_p)$) then
 3:     $((T_1,|T_1^2|_p),(T_0,|T_0^2|_p)) \Leftarrow$ RMUL($(a_0,|a_0^2|_p),(b_j,|b_j^2|_p)$)
 4:     $(T_0,|T_0^2|_p) \Leftarrow$ RADD($(T_0,|T_0^2|_p),(d_0,|d_0^2|_p)$)
 5:     $(T_1,|T_1^2|_p) \Leftarrow$ RADDC($(T_1,|T_1^2|_p),(0,0)$)
 6:     $((-,-),(U,|U^2|_p)) \Leftarrow$ RMUL($(T_0,|T_0^2|_p),(M_0',|M_0'^2|_p)$)
 7:     $((T_3,|T_3^2|_p),(T_2,|T_2^2|_p)) \Leftarrow$ RMUL($(M_0,|M_0^2|_p),(U,|U^2|_p)$)
 8:     $(-,-) \Leftarrow$ RADD($(T_0,|T_0^2|_p),(T_2,|T_2^2|_p)$)
 9:     $(T_0,|T_0^2|_p) \Leftarrow$ RADDC($(T_1,|T_1^2|_p),(T_3,|T_3^2|_p)$)
10:     $(T_1,|T_1^2|_p) \Leftarrow$ (carry, carry)
11:     for $i = 1$ to $e-1$ do
12:       if Check($(a_i,|a_i^2|_p),(b_j,|b_j^2|_p),(d_i,|d_i^2|_p),(U,|U^2|_p),(M_i,|M_i^2|_p)$) then
13:         $(T_0,|T_0^2|_p) \Leftarrow$ RADD($(T_0,|T_0^2|_p),(d_i,|d_i^2|_p)$)
14:         $(T_1,|T_1^2|_p) \Leftarrow$ RADDC($(T_1,|T_1^2|_p),(0,0)$)
15:         $((T_4,|T_4^2|_p),(T_3,|T_3^2|_p)) \Leftarrow$ RMUL($(a_i,|a_i^2|_p),(b_j,|b_j^2|_p)$)
16:         $(T_0,|T_0^2|_p) \Leftarrow$ RADD($(T_0,|T_0^2|_p),(T_3,|T_3^2|_p)$)
17:         $(T_1,|T_1^2|_p) \Leftarrow$ RADDC($(T_1,|T_1^2|_p),(T_3,|T_3^2|_p)$)
18:         $(T_2,|T_2^2|_p) \Leftarrow$ (carry, carry)
19:         $((T_4,|T_4^2|_p),(T_3,|T_3^2|_p)) \Leftarrow$ RMUL($(M_i,|M_i^2|_p),(U,|U^2|_p)$)
20:         $(d_{i-1},|d_{i-1}^2|_p) \Leftarrow$ RADD($(T_0,|T_0^2|_p),(T_3,|T_3^2|_p)$)
21:         $(T_0,|T_0^2|_p) \Leftarrow$ RADDC($(T_1,|T_1^2|_p),(T_3,|T_3^2|_p)$)
22:         $(T_1,|T_1^2|_p) \Leftarrow$ (carry, carry)
23:       else
24:         ABORT
25:       end if
26:     end for
27:     $(d_{e-1},|d_{e-1}^2|_p) \Leftarrow (T_0,|T_0^2|_p)$
28:     $(d_e,|d_e^2|_p) \Leftarrow (T_1,|T_1^2|_p)$
29:   else
30:     ABORT
31:   end if
32: end for
```

## 6 Conclusion

We have presented a novel systematic non-linear arithmetic code robust against adversarial injection of faults. Based on this code we have presented arithmetic primitives for computation with encoded digits. Integrity of the code words is assured even when overflows occur. We have further used the example of digit serial Montgomery modular multiplication to demonstrate how robust arithmetic can be deployed for fault-secure multi-precision public-key computations.

The drawback of our method is easily found. The complexity of a) predicting the joint check symbol and b) re-computing the joint check symbol from the resulting digits after each operation, will have a serious impact on the performance of our scheme. However, the reader should also keep in mind that we clearly prioritize robustness over performance. Given the increased vulnerability level of mobile and ubiquitous security devices, and the progress in adversarial fault analysis techniques, we believe this priority assessment is justified. Moore's law fill further reduce the impact that the performance gap has on the user experience.

## References

[1] R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1–11. USENIX Assoc., USENIX Press, Nov 1996.

[2] G. Gaubatz. Versatile montgomery multiplier architectures. Master's thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, May 2002.

[3] M. Karpovsky and A. Taubin. New class of nonlinear systematic error detecting codes. *IEEE Transactions on Information Theory*, 50(8):1818–1820, August 2004.

[4] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust protection against fault-injection attacks of smart cards implementing the advanced encryption standard. In L. Simoncini, editor, *Proc. Int. Conf. Dependable Systems and Networks (DSN'04)*, pages 93–101. IEEE Computer Society, IEEE Press, 2004.

[5] Ç.K. Koç, T. Açar, and B.S. Jr. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[6] K. Kulikowski, M. G. Karpovsky, and A. Taubin. Robust codes for fault attack resistant cryptographic hardware. In L. Breveglieri and I. Koren, editors, *2nd Int. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'05)*, Sep 2005.

[7] David Mandelbaum. Arithmetic codes with large distance. *IEEE Transactions on Information Theory*, 13(2):237–242, April 1967.

[8] D.K. Pradhan (ed.). *Fault Tolerant Computing – Theory and Techniques*, volume 1. Prentice-Hall, New Jersey, $1^{st}$ edition, 1986.

[9] T. R. N. Rao and O. N. Garcia. Cyclic and multiresidue codes for arithmetic operations. *IEEE Trans. Inf. Theory*, 17(1):85–91, Jan 1971.