Non-linear Residue Codes for Robust Public-Key Arithmetic

Joint work with
Gunnar Gaubatz WPI, and Mark Karpovsky Boston University

9/10/2006

Berk Sunar
Worcester Polytechnic Institute and
Visiting Researcher, Ruhr-Universitat Bochum

CRIS Lab:
http://crypto.wpi.edu

Motivation

- Boneh 1996: Via fault induction during CRT-inversion step of RSA reveals modulus factors with one simple GCD computation
- Fault induction may be facilitated to make a cryptographic IC leak secret information
- “Bellcore”-style active attacks
- Many unsubstantiated claims

Even More Motivation..

- Power balanced logic cell libraries are used to reduce the correlation between data and side-channel leakage.
- Power consumption and hence electro-magnetic emanations are data-independent, eliminates possibility of passive attacks.
- Workaround
  - *The attacker induces a fault imbalancing the power consumption,*
  - *A classical side-channel attack follows.*

Past Solutions

- Need fault detection network build right into IC.
- Previous proposals were limited to simple parity checks
- Possible solution: Linear arithmetic codes borrowed from communication theory.
  - Low overhead (<50%)
  - Assumes attacker has little control over error patterns
- Problem: There exists error vectors for which ***all*** codewords will jump to another codeword.
- Using one of these error vectors the attacker will have a high chance inserting an error that will go undetected.

A Strong Error Model

- Proposed by Karpovsky et al in FDTC 2005
- Assumptions:
  - The attacker can introduce an arbitrary number of flips in the data vectors. (has control over the weight of the error vectors).
  - Attacker may not read, compute and write on the fly. (low temporal resolution)
- Linear codes can't withstand assump. 1
- Need error checks that are data dependent.

The Error Model (cont.)

- Use code function f(x) to define code

$$C=\{ (x,w) \mid w=f(x) \}$$

and metric

$$Q(e)=|\{x \mid f(x+e_x)=f(x)+e_w,\ e\neq 0\}| \ / \ |C|$$

- The attacker has only chance max{Q(e)} to insert a error which will go undetected.
- In other words, the expected number of trial an attacker has to make to implement a successful attack is at least $\frac{1}{2}$ 1/max{Q(e)}.
- We want Q(e) to be bounded and very small for all possible e, e.g. $Q(e) < 2^{-32}$.
- The probability Q(e) of an undetected error e does not only depend on the error pattern, but also on the data itself.

A Specific Construction by Karpovsky

- Assume we are given a q-ary (q>2) linear code V(n,k) with check matrix H=[P|I] with rank(P)=n-k.
- Form the *non-linear* code

  $C_V = \{ (x,w) \mid x \in GF(q^k), w = (xG)^2 \in GF(q^r) \}$.
- Then
  - q^k-q^{k-r} errors are detected with Q(e)=0 and
  - q^n-q^k errors are detected with Q(e)=q^{-r}
- There is a similar construction for the binary char.

Practical Issues

- The non-linearity makes it difficult to implement EDN throughout the IC.
- Input /output operands in cryptographic functions rarely have such nice structures, e.g. $GF(p^k)$ or $GF((2^n)^m)$.
- Need a technique to protect arbitary datapaths (16/32/64 bits) with support for basic arithmetic operations, +/-, shifts and mul.
- End result would be protected Montgomery or Barret reduction circuits and hence protected RSA, D-H, ECC etc. designs.

A New Robust Code

- **Definition**: Let

  $C = \{ (x,w) \mid w = f(x) \in GF(p),\ x \in \mathbf{Z}_2^k \}.$

  where $f : Z_2^k \to GF(p)$ and $r = \log_2(p)$ is defined as $f(x)=x^2 \bmod p = |x^2|_p$.

- **Theorem**: C is robust if and only if $r=k$ and $2^k - p < \sigma$ where

  $\max\{Q(e)\} \leq \sigma\, 2^{-r}$

A Tight Bound on σ

- **Theorem:** Given the robust residue code C as before, the error check equation

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

  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. Hence for $e\neq 0$

  $\max\{Q(e)\} \leq 2^{-k} \max\{4, 2^k-p+1\}$

Practical Values

k	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
$2^k – p$	**1**	**5**	**1**	**3**	**9**	**3**	**15**	**3**	**39**	**5**	**39**	**57**	**3**	**35**	**1**	**5**
k	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**
$2^k – p$	**9**	**41**	**31**	**5**	**25**	**45**	**7**	**87**	**21**	**11**	**57**	**17**	**55**	**21**	**115**	**59**
k	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**
$2^k – p$	**81**	**27**	**129**	**47**	**111**	**33**	**55**	**5**	**13**	**27**	**55**	**93**	**1**	**57**	**25**	**59**

Robust coding of an arbitrary datapath

A typical datapath contains computational elements and routing elements commanded by the control logic

- Datapath width is increased to accommodate check bits
- Routing elements are not touched
- Computational elements are replaced with robust versions.
  - Need robust versions of common components
- Implement error checking/handling network
  - Self-checking checkers
  - Disable after countdown expires

Robust Addition

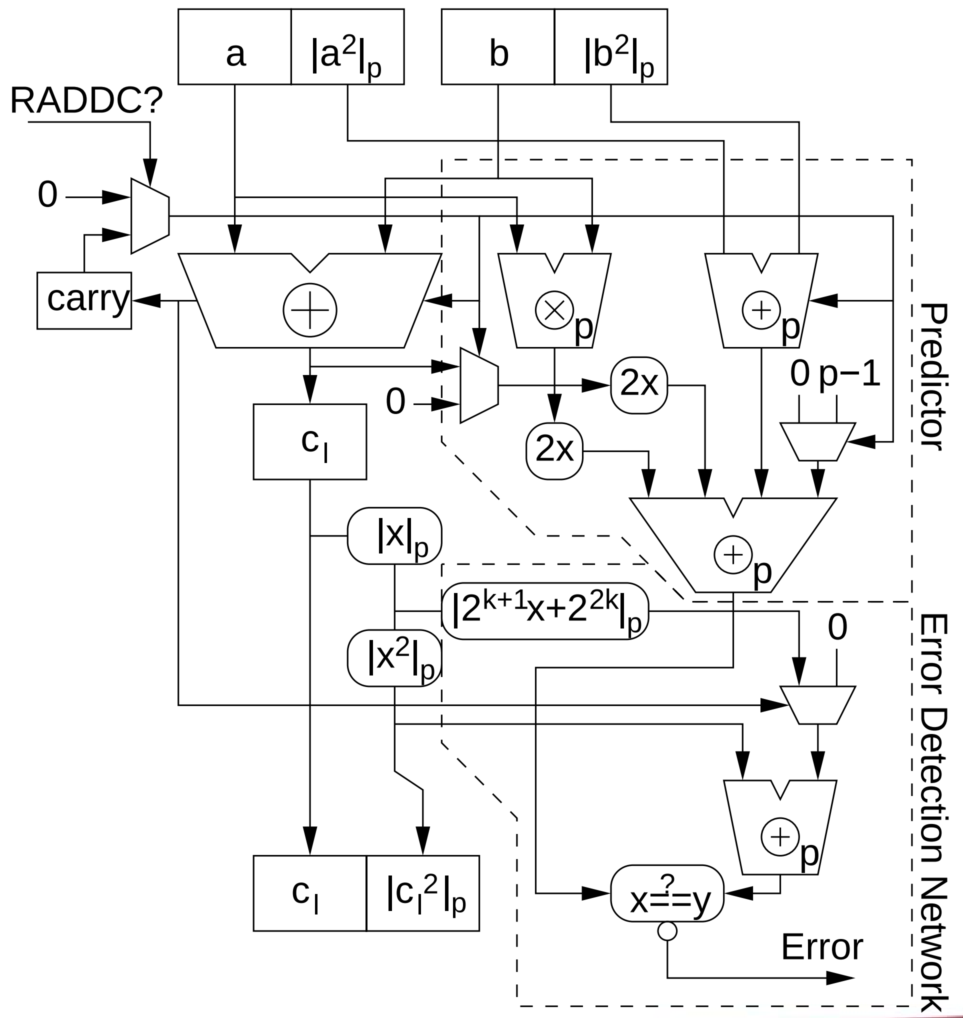
- Assume error check on operands a and b are available, e.g. $|a^2|_p$, and $|b^2|_{p\,.}$
- Need to implement ***predicted*** error check from existing error checks $|a^2|_p$, and $|b^2|_p$:

$$|c^2|_p = |(a+b+c_{in})^2|_p$$
$$= ||a^2|_p+|b^2|_p+ 2(ab+c_{in}(a+b))+c_{in}|_p$$

- Compare against ***actual*** check

$$|c^2|_p* = |(c_h2^k+c_l)^2|_p = |c_h\,|2^{2k}+c_l2^{k+1}|_p+|c_l{}^2|_p|_p$$

Robust Addition RADDC



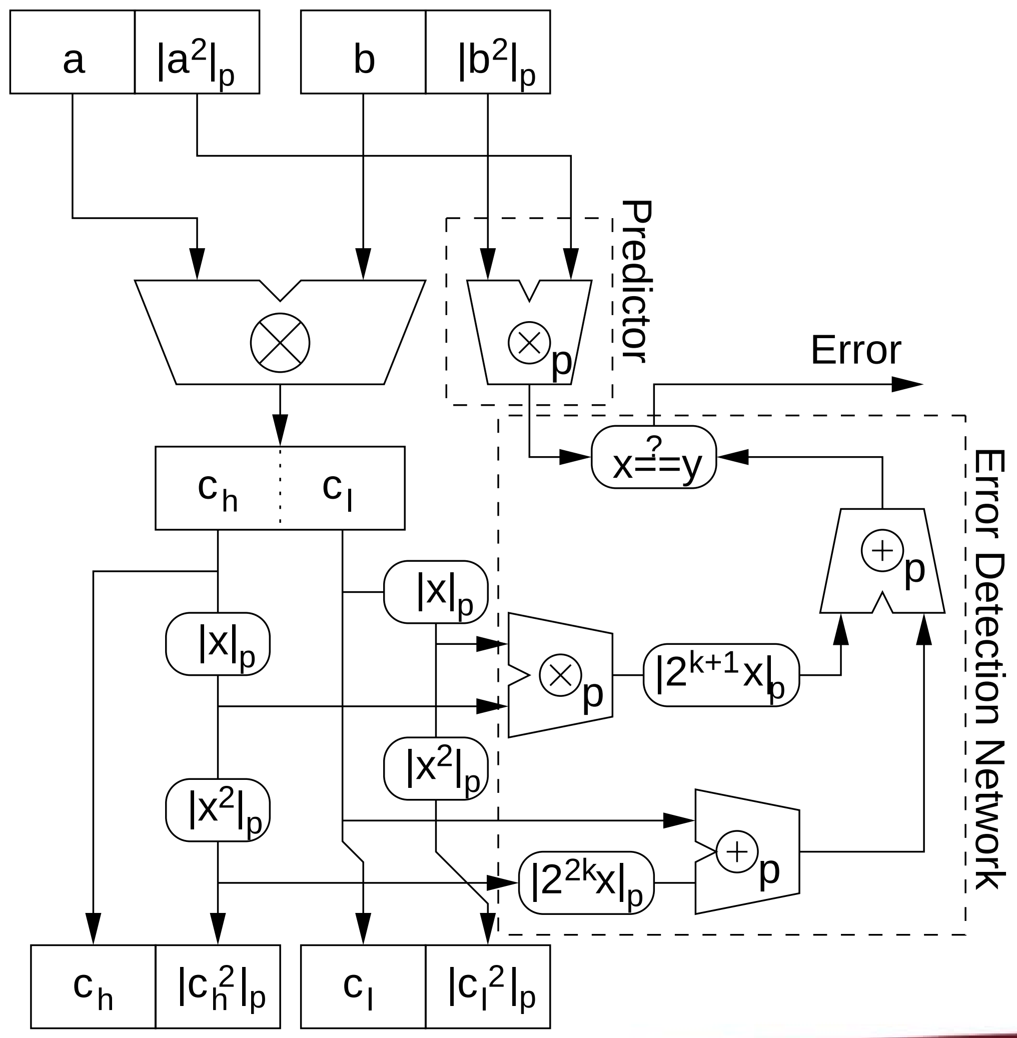
Robust Multiplication

- Given $(a, |a^2|_p)$ and $(b, |b^2|_p)$ the predicted value of the checksum is simply $|c^2|_p = |a^2|_p\, |b^2|_p$
- We compute the actual checksum of $c=ab=c_h 2^k+c_l$ as follows

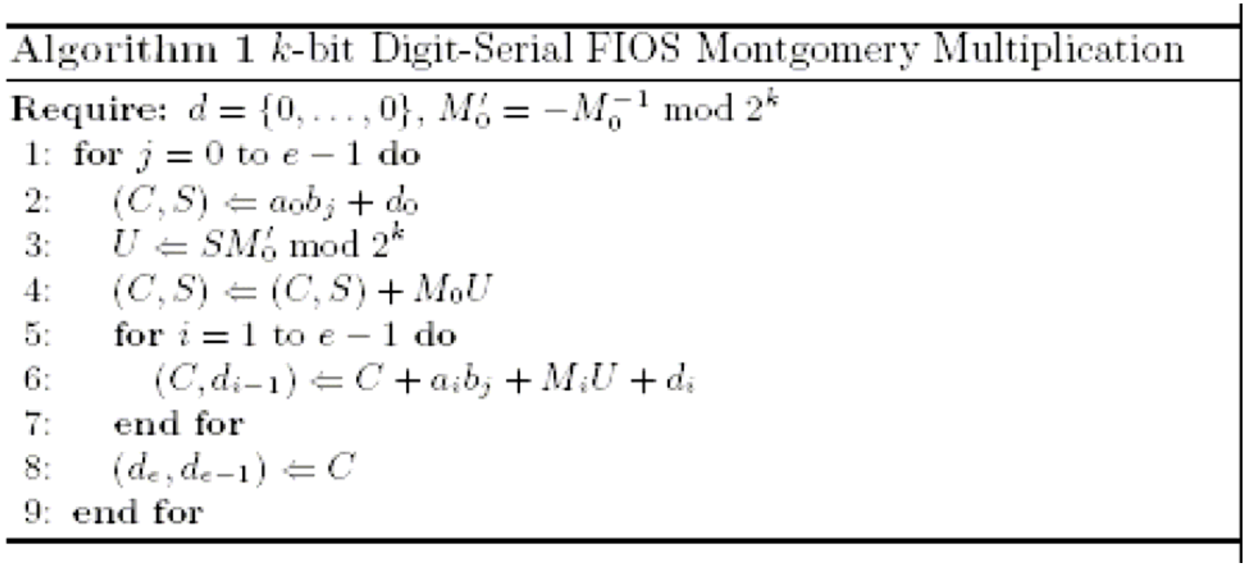
$$|c^2|_p^* = |(c_h 2^k + c_l)^2|_p$$
$$= \big|\, |c_h^2|_p\, |2^{2k}|_p + |c_h^2|_p |c_l^2|_p\, |2^{k+1}|_p + |c_l^2|_p\, \big|_p$$

- The values $|2^{2k}|_p$ and $|2^{k+1}|_p$ are constant.
- $|c_h^2|_p$ and $|c_l^2|_p$ are intermediary values of the computation which are also forwarded to the next stage of the datapath.

Robust Multiplication RMUL



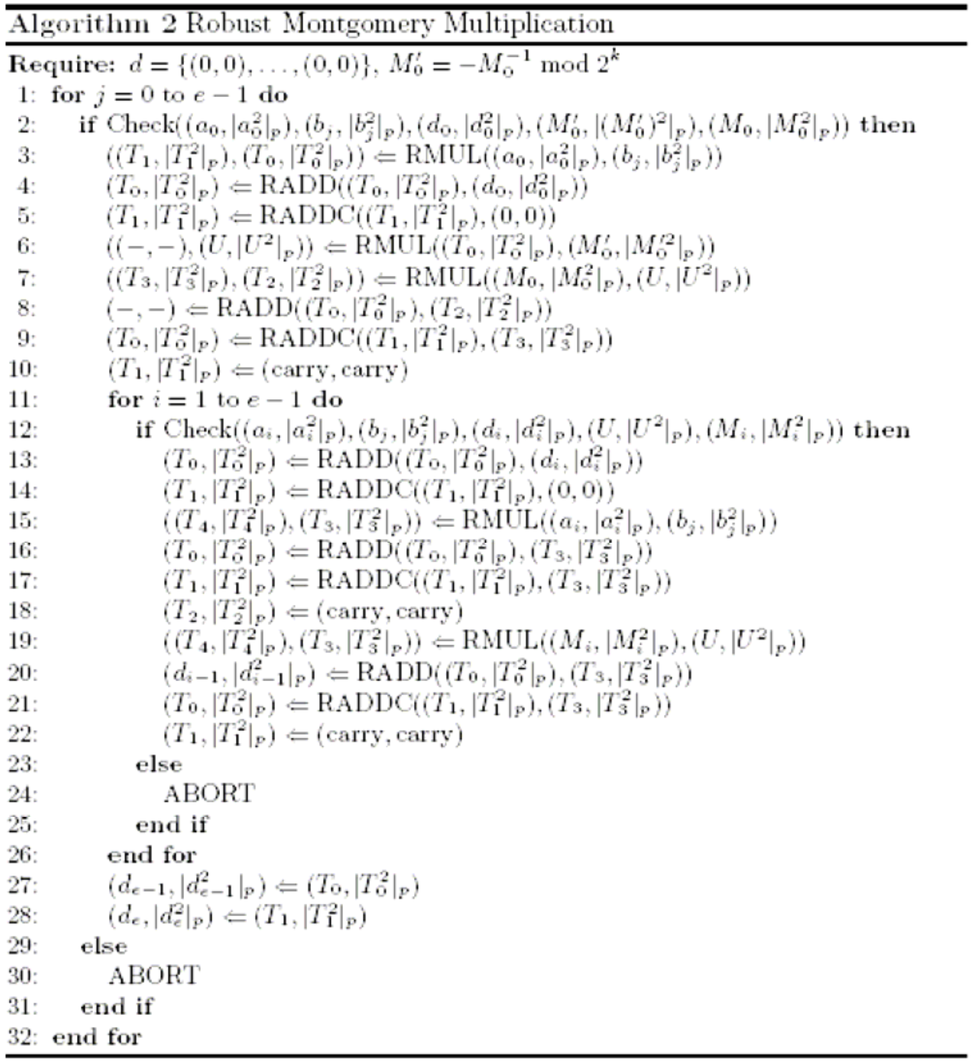
Montgomery Multiplication



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
```

Robust Montgomery Multiplication



Algorithm 2 Robust Montgomery Multiplication

```
Require: $d = \{(0,0),\ldots,(0,0)\}$, $M'_0 = -M_0^{-1} \bmod 2^k$
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:     $((T_1,|T_1^2|_p),(T_0,|T_0^2|_p)) \Leftarrow \mathrm{RMUL}((a_0,|a_0^2|_p),(b_j,|b_j^2|_p))$
4:     $(T_0,|T_0^2|_p) \Leftarrow \mathrm{RADD}((T_0,|T_0^2|_p),(d_0,|d_0^2|_p))$
5:     $(T_1,|T_1^2|_p) \Leftarrow \mathrm{RADDC}((T_1,|T_1^2|_p),(0,0))$
6:     $((-,-),(U,|U^2|_p)) \Leftarrow \mathrm{RMUL}((T_0,|T_0^2|_p),(M'_0,|M'^2_0|_p))$
7:     $((T_3,|T_3^2|_p),(T_2,|T_2^2|_p)) \Leftarrow \mathrm{RMUL}((M_0,|M_0^2|_p),(U,|U^2|_p))$
8:     $(-,-) \Leftarrow \mathrm{RADD}((T_0,|T_0^2|_p),(T_2,|T_2^2|_p))$
9:     $(T_0,|T_0^2|_p) \Leftarrow \mathrm{RADDC}((T_1,|T_1^2|_p),(T_3,|T_3^2|_p))$
10:    $(T_1,|T_1^2|_p) \Leftarrow (\mathrm{carry},\mathrm{carry})$
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:        $(T_0,|T_0^2|_p) \Leftarrow \mathrm{RADD}((T_0,|T_0^2|_p),(d_i,|d_i^2|_p))$
14:        $(T_1,|T_1^2|_p) \Leftarrow \mathrm{RADDC}((T_1,|T_1^2|_p),(0,0))$
15:        $((T_4,|T_4^2|_p),(T_3,|T_3^2|_p)) \Leftarrow \mathrm{RMUL}((a_i,|a_i^2|_p),(b_j,|b_j^2|_p))$
16:        $(T_0,|T_0^2|_p) \Leftarrow \mathrm{RADD}((T_0,|T_0^2|_p),(T_3,|T_3^2|_p))$
17:        $(T_1,|T_1^2|_p) \Leftarrow \mathrm{RADDC}((T_1,|T_1^2|_p),(T_3,|T_3^2|_p))$
18:        $(T_2,|T_2^2|_p) \Leftarrow (\mathrm{carry},\mathrm{carry})$
19:        $((T_4,|T_4^2|_p),(T_3,|T_3^2|_p)) \Leftarrow \mathrm{RMUL}((M_i,|M_i^2|_p),(U,|U^2|_p))$
20:        $(d_{i-1},|d_{i-1}^2|_p) \Leftarrow \mathrm{RADD}((T_0,|T_0^2|_p),(T_3,|T_3^2|_p))$
21:        $(T_0,|T_0^2|_p) \Leftarrow \mathrm{RADDC}((T_1,|T_1^2|_p),(T_3,|T_3^2|_p))$
22:        $(T_1,|T_1^2|_p) \Leftarrow (\mathrm{carry},\mathrm{carry})$
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
```

Performance Degradation

- Area (including check)
  - $A_{RADDC} = 2\ A_{MUL} + 4\ A_{ADD}$
  - $A_{RMUL} = 3\ A_{MUL} + 3\ A_{ADD}$
  - Both figures may be improved by coarse grain error checking
- Critical Path delay:
  - $T_{RADDC} = 1\ T_{MUL} + 1\ T_{ADD}$
  - $T_{RMUL} = 2\ T_{MUL}$
- Montgomery multiplication
  - ~3 times larger
  - ~2 times slower

Conclusion

- Further progress on new error model
- A new non-linear robust code and associated error detection scheme
- High degree of versatility (RSA, DH, ECC etc.)
- Quantifiable resilience against fault induction attacks of high precision
- Performance cost is high but can be mitigated by building specialized EDNs

Questions?

Thanks!