On Protecting Cryptographic Applications Against Fault Attacks Using Residue Codes

Kazim Yumbul
Gebze Institute of Technology
Gebze, Kocaeli, 41400 Turkey
Email: kyumbul@gyte.edu.tr

Serdar Süer Erdem
Gebze Institute of Technology
Gebze, Kocaeli, 41400 Turkey
Email: ser dem@gyte.edu.tr

***Erkay Savaş**
Sabanci University
Tuzla Istanbul, 34956 Turkey
Email: erkays@sabanciuniv.edu

FDTC 2011, 27 September, Nara Japan

OUTLINE

- Introduction
- Summary of Contribution
- Non-linear Residue Codes for Error Detection
- Adversarial Model
- Attacks on Residue Codes
- Error Detection Strategy
- Architecture
- Implementation Results
- Conclusion and Future Work

INTRODUCTION

- Adversary having physical access to cryptographic device can introduce errors during the calculations
- Faulty calculations → compromise of secret keys
- Remedy: Equip the device with *error detection* capability
- Conventional error-detection codes may not be sufficient against sophisticated attackers
- Karpovsky and Taubin proposed non-linear codes

CONTRIBUTION

- Our solution also based on non-linear codes
  - Specifically quadratic residue codes
- Our contribution:
  - Investigation of the security of quadratic residue codes against a new type of adversarial model
  - Proposal of a new residue codes using two moduli
  - Integration of the quadratic residue codes into the datapath of an embedded processor
  - Investigation of the overhead cost of the integration (chip space, time complexity)

Residue Codes for Error Detection

- Non-linear residue codes

$$C = \{(x, w) \mid x \in Z_{2k},\ w = f(x) \bmod p \in F_p\}$$

- Quadratic residue codes

$$C = \{(x, w) \mid x \in Z_{2k},\ w = x^2 \bmod p \in F_p\}$$

- Dual residue codes

$$C = \{(x, w) \mid x \in Z_{2k},\ w = f_p(x) \bmod p \,||\, f_q(x) \bmod q\ \}$$

$$w = w_p//w_q,\ w_p = f_p(x) \bmod p\ and\ w_q = f_q(x) \bmod q$$

Undetected Errors

- Let e_x and e_w denote errors in the data x and parity w, respectively
- Undetected errors

$$f(x + e_x \bmod 2^k) \bmod p = w + e_w \, mod \, 2^r\}$$

- The probability that this error remains undetected (error masking probability)

$$Q(e) = Q(e_x, e_w) = \frac{\left|\{x \mid (x + e_x, w + e_w) \in C\}\right|}{|C|}$$

Adversarial Model

- Assumptions on the (*powerful*) adversary
  - cannot read the bits in the data path; i.e. x and w are unknown.
  - can flip bits of the data x and the parity w to generate undetectable errors
- Example: simple residue code
  - $w = x \bmod p$, where $p = 2^r - 1$ and $r < k$
  - A data word $x = (x_{k-1}, \ldots, x_r, x_{r-1}, \ldots x_1, x_0)$
  - Attack: $x_m = (x_{k-1}, \ldots, x_r', x_{r-1}, \ldots x_1, x_0')$
  - If $x = (x_{k-1}, \ldots, 0, x_{r-1}, \ldots x_1, 1) \rightarrow x_m = x + p \rightarrow w = x_m \bmod p$

Attacking Quadratic Residue Codes

- Security depends on the choice of the modulus
- Example:
  - $p = 2^{32}-5$ (suitable for protecting computer words)
  - $p = 11111111111111111111111111111011$
  - Data words of the form $x = (x_{31},\ldots, x_4, x_3, 0, x_1, x_0)$
  - Attack: $x_m = (x_{31}',\ldots, x_4', x_3', 0, x_1', x_0')$
  - $x_m = p\text{-}x$ ➔ $w = x^2 \bmod p = (p\text{-}x)^2 \bmod p$
  - Success probability: 50%
- A better modulus may result in poor implementation
  - $p = \text{0xFB01CDD9}$

Dual Residue Codes

- Basic idea is to use two moduli, p and q
- Parity
  - $w = f_p(x) \bmod p \,\|\, w_q = f_q(x) \bmod q$
- Attacking Example:
  - $w_p = x \bmod 2^{19}\text{-}1$ and $w_q = x^2 \bmod 2^{13}\text{-}1$
  - $x = (x_{31}, \ldots, 0, x_{12}, \ldots, x_1, 1) \rightarrow x_m = x + q \rightarrow w_q = x_m \bmod q$
  - $w_p \neq x_m \bmod p = x \bmod p + 2^{13}\text{-}1 \bmod p$
  - If $w_p = (w_{p,18}, \ldots, 0, w_{p,12}, \ldots, w_{p,1}, 1)$
  - Success rate is 1/16 if we are able to flip the bits x_{13}, x_0, $w_{p,13}$, and $w_{p,0}$.

Quadratic Dual Residue Codes

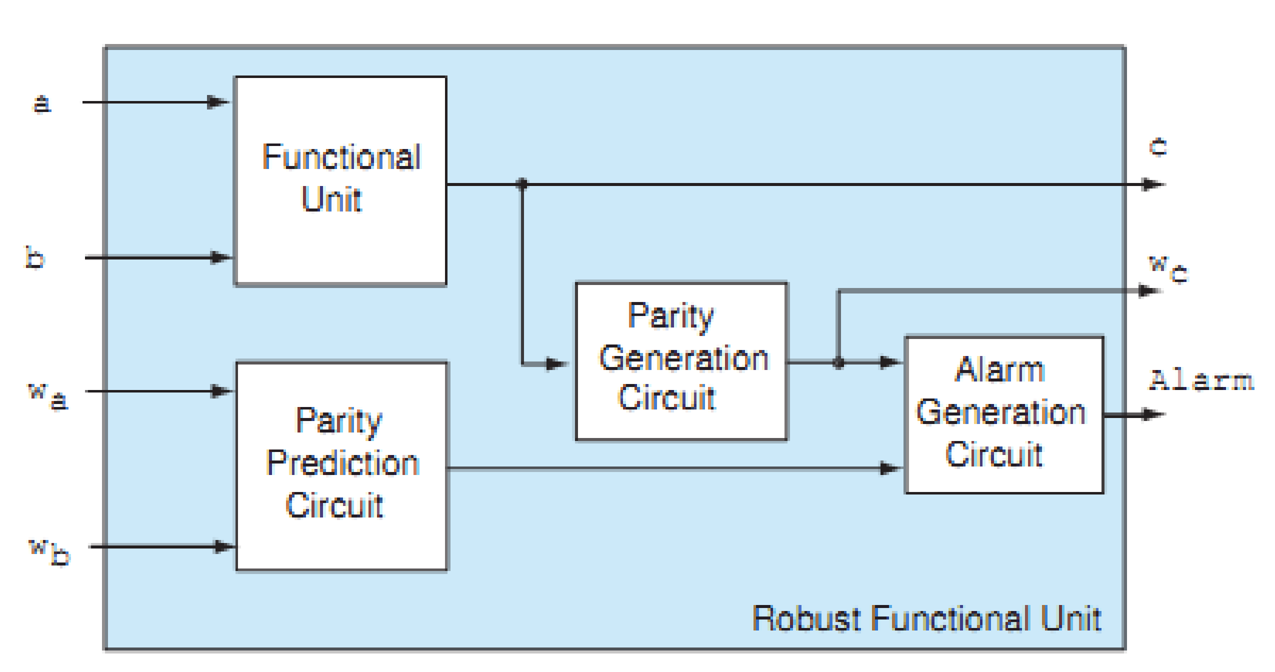
- Definition
  - $w_p = x^2 \bmod p$ and $w_q = x^2 \bmod q$.
- Due to Chinese Remainder Theorem
  - there are four data words x_1, x_2, x_3, x_4 that have the same parity
  - $s_p = x \bmod p$ and $s_q = x \bmod q$
  - $x_1 \equiv (s_p \cdot N_p \cdot M_p + s_q \cdot N_q \cdot M_q) \bmod n$
  - $x_2 \equiv (-s_p \cdot N_p \cdot M_p + s_q \cdot N_q \cdot M_q) \bmod n$
  - $x_3 \equiv (s_p \cdot N_p \cdot M_p - s_q \cdot N_q \cdot M_q) \bmod n$
  - $x_4 \equiv (-s_p \cdot N_p \cdot M_p - s_q \cdot N_q \cdot M_q) \bmod n$

Quadratic Dual Residue Codes

- Attack scenarios
  - $x_1 \equiv - x_4 \bmod n$
    - $p = 2^{19}$-1 and $q = 2^{13}$-1
    - $n = p \cdot q = 111111111111011111100000000000001$ ➔ difficult
  - $x_1 \equiv x_2 \bmod q$
  - $x_1 \equiv - x_2 \bmod p$
- Adding a multiple of n *to* x.
  - Since $n < 2^{32}$-1 it is possible, however difficult and unlikely
  - Choose $k = 31$ ➔ some performance implications

Robust Functional Unit

- Input parities are known
  - Since they are output of other robust functional unit



Robust Adder

Predicted parity

$$w_c^* = \left|(a+b+c_{in})^2\right|_p$$
$$= \left|\left|a^2\right|_p + \left|b^2\right|_p + 2(ab + c_{in}(a+b)) + c_{in}\right|$$

Calculated parity

$$w_c = \left| \ (c_H.2^k + c_L)^2 \ \right|_p$$
$$= \left| \ c_H.\left|2^{2k}\right|_p + c_H.\left|c_L\right|_p.\left|2^{k+1}\right|_p + \left|c_L^2\right|_p \ \right|_p$$
$$= \left| \ c_H.\left|2^{2k} + c_L.2^{k+1}\right|_p + \left|c_L^2\right|_p \ \right|_p$$

Check

$$w_c = w_c^*.$$
$$\left| \ c_H.\left|2^{2k} + c_L.2^{k+1}\right|_p + \left|c_L^2\right|_p \ \right|_p =$$
$$w_a + w_b + \left|2(ab + c_{in}(a+b)) + c_{in} \ \right|_p$$

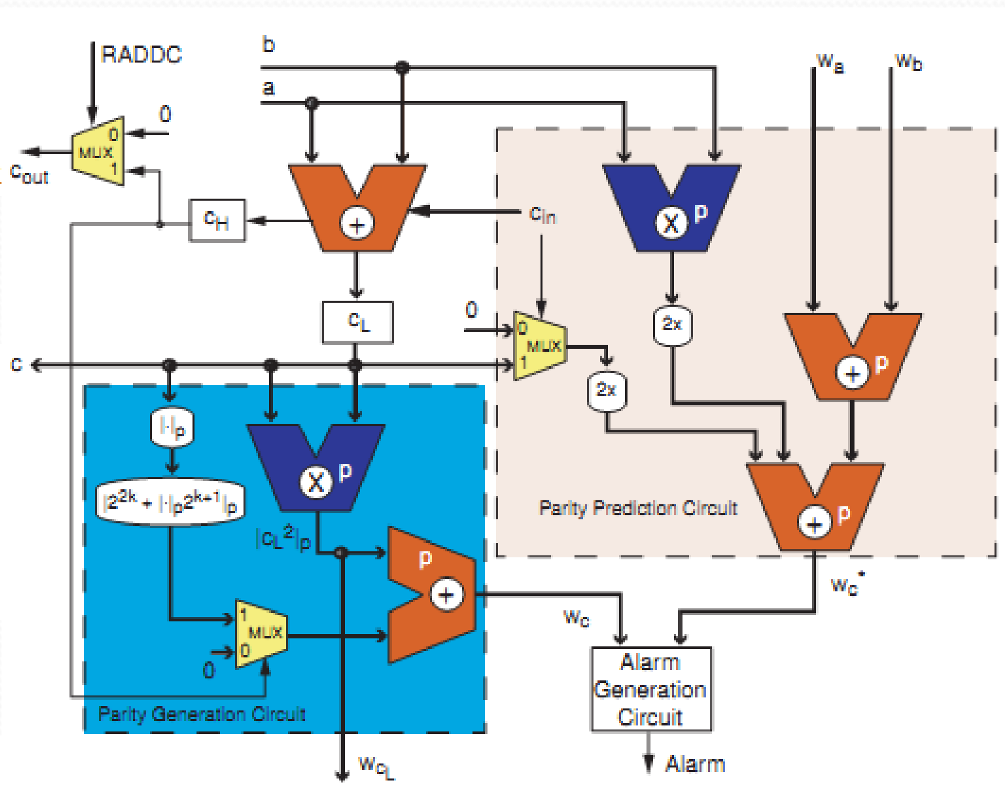


Figure 4. Robust Addition Unit

Robust Multiplier

Predicted parity

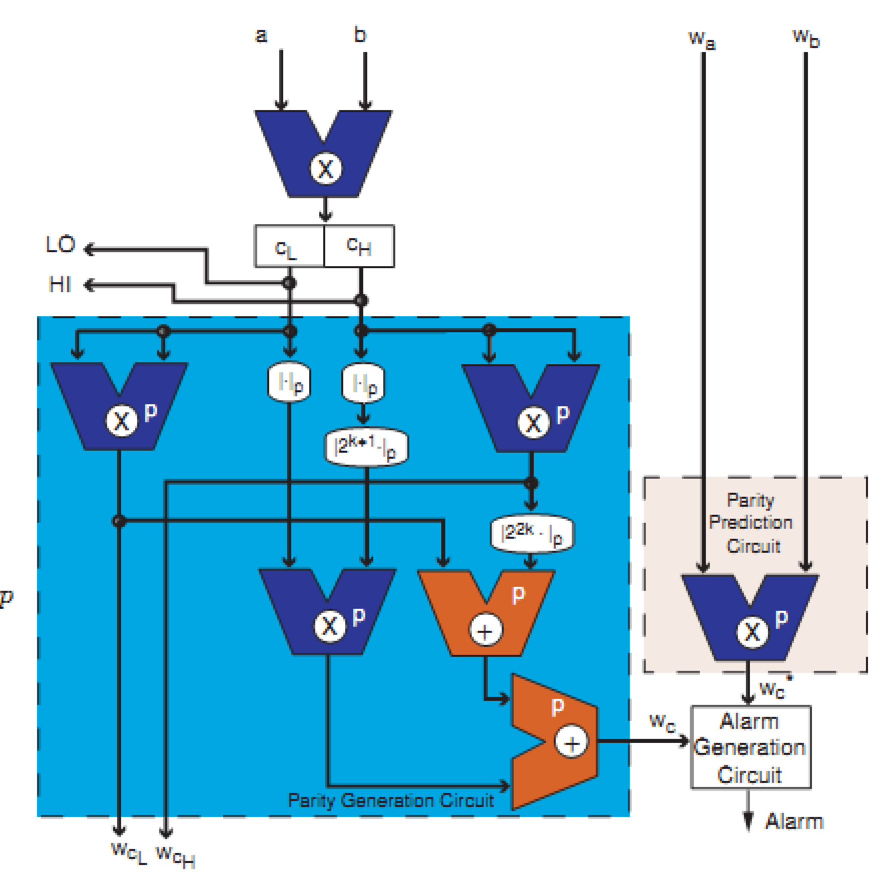
$$w_c^* = \left| \, |a^2|_p \cdot |b^2|_p \, \right|_p$$
$$= |w_a \cdot w_b|_p$$

Calculated parity

$$w_c = \left| \, (c_H.2^k + c_L)^2 \, \right|_p$$
$$= \left| \, c_H^2.2^{2k} + c_H.c_L.2^{k+1} + c_L^2 \, \right|_p$$
$$= \left| \, |c_H^2|_p.|2^{2k}|_p + |c_H|_p.|c_L|_p.|2^{k+1}|_p + |c_L^2|_p \, \right|_p$$

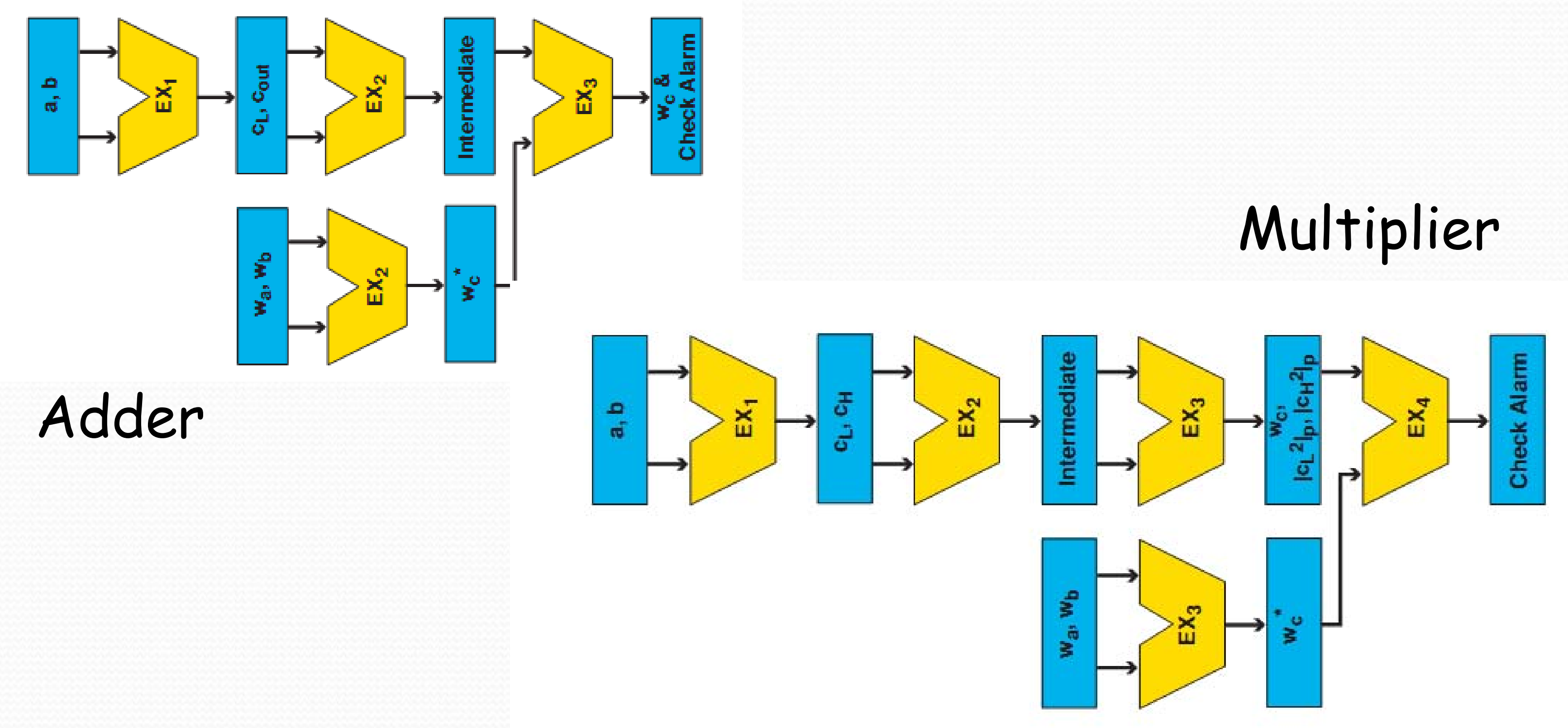
Check

$$w_c = w_c^*.$$



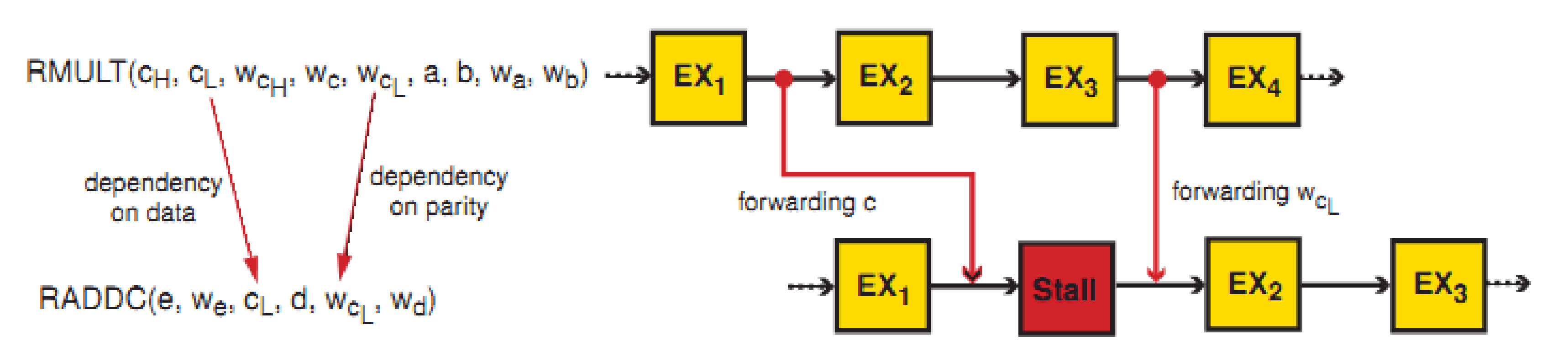
Integration

- Pipeline Integration of the Proposed Robust Functional Units



Data Hazards

- Occasional pipeline stalls
  - Due to data dependencies and long lasting robust operations
  - Reordering can eliminate most.



Processor Configurations

- Configuration 0:
  - 32-bit Xtensa LX3 microprocessor
  - A simple embedded processor without robust units
- Configuration 1:
  - $p = 2^{32} - 5$ ($w_p = x^2 \bmod p$) and k = 32
  - Unsafe against the proposed adversary model
- Configuration 2:
  - $p = 2^{31} - 1$ ($w_p = x^2 \bmod p$) and k = 31
  - Unsafe against the proposed adversary model
  - Easier to implement in hardware

Processor Configurations

- Configuration 3
  - $w_p = x^2 \bmod p$, $w_q = x \bmod q$
  - $p = 2^{19} - 1$ and $q = 2^{13} - 1$ ($k = 32$)
  - Unsafe against the proposed adversary model
- Configuration 4
  - $w_p = x^2 \bmod p$, $w_q = x \bmod q$
  - $p = 2^{19} - 1$ and $q = 2^{13} - 1$ ($k = 31$)
  - Unsafe against the proposed adversary model

Processor Configurations

- Configuration 5
  - $w_p = x^2 \bmod p$, $w_q = x^2 \bmod q$
  - p = $2^{19} - 1$ and q= $2^{13} - 1$ ($k = 32$)
  - Good protection against the proposed adversary model
- Configuration 6
  - $w_p = x^2 \bmod p$, $w_q = x^2 \bmod q$
  - p = $2^{19} - 1$ and q= $2^{13} - 1$ ($k = 31$)
  - Good protection against the proposed adversary model

Implementation Results

Lower clock count due to optimized functional units

Table I
CLOCK CYCLE COMPARISON FOR MONTGOMERY IMPLEMENTATION

Configurations	2048-bit	1024-bit	512-bit
Configuration 0	304, 585	88, 588	29, 859
Configuration 1	129, 802	36, 186	12, 394
Configuration 2	208, 571	57, 936	18, 126
Configuration 3	190, 791	51, 845	16, 592
Configuration 4	208, 564	57, 894	18, 119
Configuration 5	190, 791	51, 845	16, 592
Configuration 6	208, 564	57, 894	18, 119

~ 50% increase in ASIC area

Table II
SPEED AND AREA INFORMATION FOR ASIC IMPLEMENTATION

Configurations	CPU Speed (MHz)	Base CPU Area	TIE Area	Total Area
Configuration 0	320	64, 000	0	64, 000
Configuration 1	320	64, 000	33, 076	97, 076
Configuration 2	320	64, 000	32, 985	96, 985
Configuration 3	320	64, 000	32, 300	96, 300
Configuration 4	320	64, 000	32, 289	96, 289
Configuration 5	320	64, 000	34, 918	98, 918
Configuration 6	320	64, 000	34, 912	98, 912

Implementation Results

Negligible decrease in clock frequency

Less than 50% increase

more DSP units

Table III
SPEED AND GATE INFORMATION FOR FPGA IMPLEMENTATION WITH TIME CONSTRAINT @ 33.33 MHz

Configurations	Max Clock Frequency (MHz)	Slice Count	LUT Count	RAM16b	DSP48s
Configuration 0	37.281	7, 751	19, 958	272	1
Configuration 1	33.338	9, 263	26, 761	272	1
Configuration 2	34.338	9, 395	26, 701	272	0
Configuration 3	36.004	8, 917	26, 082	272	2
Configuration 4	36.004	8, 915	26, 080	272	2
Configuration 5	33.940	8, 235	26, 202	272	4
Configuration 6	33.940	8, 234	26, 198	272	4

Table IV
SPEED AND GATE INFORMATION FOR FPGA IMPLEMENTATION WITH NO TIME CONSTRAINT

Configurations	Max Clock Frequency (MHz)	Slice Count	LUT Count	RAM16b	DSP48s
Configuration 0	57.621	7, 751	19, 958	272	1
Configuration 1	34.338	9, 263	26, 761	272	1
Configuration 2	40.538	9, 395	26, 701	272	0
Configuration 3	45.460	8, 917	26, 082	272	2
Configuration 4	45.460	8, 915	26, 080	272	2
Configuration 5	39.001	8, 235	26, 202	272	4
Configuration 6	39.001	8, 234	26, 198	272	4

Some decrease in clock frequency

Conclusion and Future Work

- Certain residue codes are shown to be insecure in the adopted adversarial model
- A new class of error detection codes is proposed
- Robust functional units utilizing residue codes are designed and implemented in an embedded processor
- Implementation results of the new processor core for both ASIC and FPGA are reported

Conclusion and Future Work

- The results show that it is possible to incorporate powerful error detection circuitry even into an embedded processor core if low to moderate increases in area and time are tolerable.
- The adopted error detection strategy benefits many cryptographic applications that uses basic arithmetic operations.
- Need for more analysis of the residue codes
  - We already obtained new results

QUESTIONS ?

THANK YOU