Cryptographic applications

Kris Gaj
Tarek El-Ghazawi
& GMU, GWU students
Cipher

message

K bits

ciphertext

cryptographic key
Definition of a cipher

**Enciphering transformation** \( E_K(M) \)

- **Message space**: \( M \)
- **Key space**: \( K \)
- **Ciphertext space**: \( C \)

**Deciphering transformation** \( D_K(C) \)

\[
\forall K \in K \quad \forall M \in \hat{M} \quad D_K(E_K(M)) = M
\]
Substitution Cipher

Key = \[
\begin{bmatrix}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} & \text{i} & \text{j} & \text{k} & \text{l} & \text{m} & \text{n} & \text{o} & \text{p} & \text{q} & \text{r} & \text{s} & \text{t} & \text{u} & \text{v} & \text{w} & \text{x} & \text{y} & \text{z} \\
\text{f} & \text{q} & \text{i} & \text{s} & \text{h} & \text{n} & \text{c} & \text{v} & \text{j} & \text{t} & \text{y} & \text{a} & \text{u} & \text{w} & \text{d} & \text{r} & \text{e} & \text{x} & \text{l} & \text{b} & \text{m} & \text{z} & \text{o} & \text{g} & \text{k} & \text{p}
\end{bmatrix}
\]

enciphering

\[
\begin{align*}
\text{TO} & \quad \text{BE} & \quad \text{OR} & \quad \text{NOT} & \quad \text{TO} & \quad \text{BE} \\
\downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
\text{BD} & \quad \text{QH} & \quad \text{DX} & \quad \text{WDB} & \quad \text{BD} & \quad \text{QH}
\end{align*}
\]

deciphering

\[
\begin{align*}
\text{TO} & \quad \text{BE} & \quad \text{OR} & \quad \text{NOT} & \quad \text{TO} & \quad \text{BE} \\
\downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
\text{TO} & \quad \text{BE} & \quad \text{OR} & \quad \text{NOT} & \quad \text{TO} & \quad \text{BE}
\end{align*}
\]

Number of keys = 26! \approx 4 \cdot 10^{26}
Secret-key (Symmetric) Cryptosystems

key of Alice and Bob - $K_{AB}$

Network

 Encryption

 Alice

 Decryption

 Bob
Key Distribution Problem

\[ \text{Users} \quad \Longrightarrow \quad \frac{N \cdot (N-1)}{2} \quad \text{Keys} \]

<table>
<thead>
<tr>
<th>Users</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5,000</td>
</tr>
<tr>
<td>1000</td>
<td>500,000</td>
</tr>
</tbody>
</table>
Public Key (Asymmetric) Cryptosystems

Public key of Bob - $K_B$

Private key of Bob - $k_B$

Network

Encryption

Decryption

Alice

Bob
Key exchange for secret-key ciphers

Network

Alice  session key  (random secret-key)  Bob
Bob’s public key

Bob’s private key

Session key encrypted using Bob’s public key

Message encrypted using session key
Digital Signature

Alice

Message

Hash function

Hash value

Public key cipher

Signature

Alice’s private key

Bob

Message

Hash function

Hash value 1

Hash value 2

Public key cipher

Signature

Alice’s public key
High-throughput secret-key encryption
High-Throughput Encryption

\[ \begin{align*}
M_i & \\
M_{i+1} & \\
M_{i+2} & \\
\end{align*} \]

\[ \begin{align*}
C_i & \\
C_{i+1} & \\
C_{i+2} & \\
\end{align*} \]

Encryption algorithms:

- DES
- 3DES
- AES
- RC5
- IDEA
- etc.

K_0
Fully Pipelined Architecture

- Loop unrolling
- Pipeline stages inside of cipher rounds
- New input & new output every clock cycle
Secret-key cipher breaking
Secret-key cipher breaking

**Given:**
- ciphertext
- guessed fragment of the plaintext

**Looked for:**
- remaining plaintext
  - or key

**Method:**
- exhaustive key search (brute-force) attack
  - successive keys
  - cipher
Secret-key breaking

Message – Ciphertext pair

\[
\begin{array}{c}
\text{Message} \\
M_0
\end{array}
\quad
\begin{array}{c}
\text{Ciphertext} \\
C_0
\end{array}
\]

Negligibly small input/output

Huge amount of computations

Cipher breaker

Correct key

\[
\begin{array}{cccc}
K_1 & K_2 & K_3 & K_N
\end{array}
\]

Generated by the cipher breaker
Public-key cryptography for key exchange and digital signatures
RSA public-key encryption

message $M$ $\rightarrow$ Public Key $\rightarrow$ ciphertext $C = f(M) = M^e \mod N$

$M = f^{-1}(C) = C^d \mod N$ $\leftarrow$ Private Key

$N = P \cdot Q$ $\quad$ $P, Q$ - large prime numbers

e \cdot d \equiv 1 \mod ((P-1)(Q-1))$
RSA keys

PUBLIC KEY

\{ e, N \} \quad \rightarrow \quad \{ d, P, Q \} \quad \leftarrow \quad \{ e, N \}

PRIVATE KEY

P, Q: P, Q - large prime numbers

N: \quad N = P \cdot Q

e: \quad \gcd(e, P-1) = 1 \quad \text{and} \quad \gcd(e, Q-1) = 1

d: \quad e \cdot d \equiv 1 \mod ((P-1)(Q-1))
Why public-key cryptography is a good application for reconfigurable computers?

• computationally intensive arithmetic operations

• unconventionally long operand sizes (160-2048 bits)

• multiple algorithms, parameters, key sizes, and architectures = need for reconfiguration
Elliptic Curve Cryptosystems (ECC)

☑ a family of cryptosystems, rather than a single cryptosystem = added security but need for reconfiguration
☑ public key (asymmetric) cryptosystems used for key agreement and digital signatures
☑ implementations must be optimized for minimum latency rather than maximum throughput = limited speed-up from parallel processing
Classes of applications

1. input/output intensive applications
   • bulk data encryption
     (DES, IDEA, and RC5 encryption)

2. computationally intensive applications
   • secret-key cipher breaking based on the exhaustive key search
     (DES, IDEA, RC5 breakers)
   • public-key cipher breaking based on factoring

3. latency-critical applications
   • cipher key agreement and signature
     (ECC schemes, RSA)
Secret key cipher libraries

1. Secret key cipher encryption and decryption
   • DES
   • IDEA
   • RC5

2. Secret key cipher breaking
   • DES
   • IDEA
   • RC5
Public key cipher libraries

1. Operations in the binary Galois Fields GF($2^m$)
   
   a. polynomial basis
   b. normal basis

2. Multiprecision integer arithmetic

3. Elliptic Curve Operations
   
   - addition
   - doubling
   - scalar multiplication
Basic operations of ECC

Basic operations in Galois Field GF(2^m)
- addition and subtraction (xor): x+y, x-y (XOR)
- multiplication, squaring: x ⋅ y, x^2
- inversion: x^{-1}

Basic operations on points of an Elliptic Curve
- addition of points: P + Q
- doubling a point: 2P
- projective to affine coordinate: P2A

Complex operations on points of an Elliptic Curve
- scalar multiplication: k ⋅ P = P + P + …+P
  \[ \text{k times} \]
Hierarchy of functions

High level

Medium level

Low level 2

Low level 1

kP

P+Q

2P

projective_to_affine (P2A)

INV

XOR

MUL

ROT
SRC Program Partitioning

µP system

\{ C function for \mu P \}

\{ C function for MAP \}

FPGA system

\{ VHDL macro \}

HLL

HDL
Investigated Partitioning Schemes
μP Software Only

C function for μP

C function for FPGA

VHDL macro

Based on public-domain code by Rosing M., *Implementing Elliptic Curve Cryptography*, Manning, 1999
0HL1 Partitioning

C function for μP

C function for FPGA

VHDL macros

MUL4  ROT  XOR  MUL2  MUL  VAR ROT

2P  P+Q  kP  P2A  INV

L1  H  0
0HL2 Partitioning

C function for μP

C function for FPGA

VHDL macros

MUL4 | ROT | XOR | MUL2 | INV

kP

2P | P+Q | P2A

L2

30
0HM Partitioning

C function for μP

C function for FPGA

VHDL macros

\[ kP \]

\[ P + Q \]

\[ 2P \]

\[ P2A \]
00H Partitioning (VHDL only)

C function for μP

C function for FPGA

VHDL macro

\[ kP \]

H
Results
Timing Measurements

MAP Alloc.

MAP function

FPGA Configure

DMA Data In

FPGA Computation

DMA DataOut

MAP Free

End-to-End time (HW)

End-to-End time (SW)

MAP Allocation time

Configuration time

MAP Release Time

.c file

.mc file
### Results (Latency)

<table>
<thead>
<tr>
<th>System Level Architecture</th>
<th>End-to-End Time (µs)</th>
<th>Data Transfer In Time (µs)</th>
<th>Data Computation Time (µs)</th>
<th>Data Transfer Out Time (µs)</th>
<th>Total Overhead (µs)</th>
<th>Speedup vs. Software</th>
<th>Slowdown vs. VHDL macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>772,519</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0HL1</td>
<td>866</td>
<td>37</td>
<td>472</td>
<td>14</td>
<td>394</td>
<td>893</td>
<td>1.46</td>
</tr>
<tr>
<td>0HL2</td>
<td>863</td>
<td>37</td>
<td>469</td>
<td>14</td>
<td>394</td>
<td>895</td>
<td>1.45</td>
</tr>
<tr>
<td>0HM</td>
<td>592</td>
<td>37</td>
<td>201</td>
<td>12</td>
<td>391</td>
<td>1305</td>
<td>1</td>
</tr>
<tr>
<td>VHDL macro</td>
<td>592</td>
<td>39</td>
<td>201</td>
<td>17</td>
<td>391</td>
<td>1305</td>
<td>1</td>
</tr>
</tbody>
</table>

![Bar Chart](chart.png)

- **End to End Time**
- **FPGA Computation Time**

**Different Architectures:**

- 0HL1
- 0HL2
- 0HM
- VHDL macro

**Usual Values:**

- 0
- 100
- 200
- 300
- 400
- 500
- 600
- 700
- 800
- 900

**Us Values:**

- 0
- 35

**Us Units:**

- µs
# Results (Area)

<table>
<thead>
<tr>
<th>System Level Architecture</th>
<th>% of CLB slices (out of 33792)</th>
<th>CLB increase vs. pure VHDL</th>
<th>% of LUTs (out of 67,584)</th>
<th>LUT increase vs. pure VHDL</th>
<th>% of FFs (out of 67,584)</th>
<th>FF count increase vs. pure VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>0HL1</td>
<td>99</td>
<td>1.68</td>
<td>57</td>
<td>1.3</td>
<td>68</td>
<td>2.61</td>
</tr>
<tr>
<td>0HL2</td>
<td>92</td>
<td>1.56</td>
<td>52</td>
<td>1.18</td>
<td>62</td>
<td>2.38</td>
</tr>
<tr>
<td>0HM</td>
<td>75</td>
<td>1.27</td>
<td>48</td>
<td>1.09</td>
<td>39</td>
<td>1.5</td>
</tr>
<tr>
<td>00H</td>
<td>59</td>
<td>1</td>
<td>44</td>
<td>1</td>
<td>26</td>
<td>1</td>
</tr>
</tbody>
</table>

![Bar Chart](image)
## Number of lines of code

<table>
<thead>
<tr>
<th>Algorithm Partitioning Scheme</th>
<th>VHDL</th>
<th>Macro Wrapper</th>
<th>MAP C</th>
<th>Main C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0HL1</td>
<td>1007</td>
<td>260</td>
<td>371</td>
<td>153</td>
</tr>
<tr>
<td>0HL2</td>
<td>1291</td>
<td>230</td>
<td>349</td>
<td>153</td>
</tr>
<tr>
<td>0HM</td>
<td>1744</td>
<td>160</td>
<td>185</td>
<td>153</td>
</tr>
<tr>
<td>VHDL macro</td>
<td>1960</td>
<td>36</td>
<td>78</td>
<td>153</td>
</tr>
</tbody>
</table>
Conclusions
Conclusions

Assuming focus on:

- Timing
- Resources
- Ease of programming
Conclusions – cont.

The best implementation approach:

0HL1 partitioning scheme

893 speedup vs. software and only 0.46 times slowdown versus pure VHDL with ease of implementation
Conclusions

• Elliptic Curve Cryptosystem implementation challenging for reconfigurable computers because of
  • optimization for latency rather than throughput
  • limited amount of parallelism

• First publication showing a 1000x speed-up for a reconfigurable computer application optimized for data latency
General hierarchy of library files suggested by SRC Computers Inc.
Structure of the macro repository

< top of repository >
|< macros >

<lib # 1 >
|<lib # 2 >
|<lib # 3 >

common
| rev_d
| rev_e
| rev_f

| macro1
| macro2
| macro3

| hdlfile
| InfoFile
| BlkBoxFile
| DebugCodeFile
| DataSheet
Files describing the macro

Platform independent
  – HDL file: macro.v or macro.vh
    • Verilog or VHDL code defining the macro
  – Debug Code File: macro.c
    • provides the equivalent C functionality for the macro
  – Data sheet file: datasheet
    • contains the documentation for the macro

Platform dependent
  – Blk Box File: blackbox.v
    • Interface (black box) definition for the macro in Verilog
  – Info File: info
    • Info file entry for this macro
Software libraries and their role in the development of SRC libraries
Roles of software libraries

1. source of test vectors for VHDL macros

2. emulation of hardware in the debug mode

3. performance comparison
How to approach porting your application to reconfigurable computers?

1. Identify class of applications

2. Identify basic operations required by your applications

3. Determine the existence of the RC library of such operations

4. Determine the existence of the microprocessor library of such operations

5. Determine the right granularity for the required library operations