Implementing ChaCha Based Crypto Primitives on Programmable SmartNICs

Shaguftha Zuveria Kottur
Krishna Kadiyala, Praveen Tammana, and Rinku Shah

ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures (FFSPIN)
August 22, 2022
Datacenter control applications offloaded to PDPs

PDP - Programmable Data Plane
Datacenter control applications offloaded to PDPs

- reduce latency
- increase server CPU savings
Example: Dispatcher in Serverless computing

Dispatcher in Serverless computing

Secured using gRPC based TLS
Dispatcher in Serverless computing - Offloaded

Kernel Network Stack

Load Balancer
Dispatcher
Hardware

Server0

Load Balancer
Dispatcher
Hardware

Server1

Worker Node
Dispatcher in Serverless computing - Offloaded
SSL/TLS works in user space. How to secure offloaded communication?
Other offloaded applications

- Failover Manager\(^2\)
- Control & Management Applications
- Replication Manager\(^1,2\)
- Consensus\(^1,5\)
- Congestion-aware load balancing\(^3\)
- Distributed Transactions\(^1\)
- Host In-band Network Telemetry\(^4\)

Existing in-network crypto processing solutions

Accelerators
- AES accelerators
  - Nvidia Bluefield, Pensando DSC
- TLS handshake offload to Nvidia Bluefield

Offloads
- AES/GCM offload to Mellanox ASIC NICs
- AES offload to Intel Tofino
  - using Scrambled Lookup Tables

Are there other cipher suites?

TLS 1.3 supports TWO ciphersuites

- AES - GCM
- ChaCha20 - Poly1305

ChaCha stream cipher

- Processor friendly Add-Rotate-XOR operations
- Resistant to side channel cache timing attacks\(^1\)

---

Key idea

Offload ChaCha based crypto primitives to smartNIC without using accelerators/co-processors
Key contributions

1. Identification of applications that benefit from offloaded crypto primitives

2. Implementation of ChaCha based crypto primitives on Netronome smartNIC

3. Performance evaluation of proposed implementation
ChaCha Overview
### ChaCha Stream Cipher: State Initialization

Increment for each 512 bit of pkt

<table>
<thead>
<tr>
<th>32b</th>
<th>32b</th>
<th>32b</th>
<th>32b</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>constant</td>
<td>constant</td>
<td>constant</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>key</td>
<td>key</td>
<td>key</td>
<td>key</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>key</td>
<td>key</td>
<td>key</td>
<td>key</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>counter</td>
<td>nonce</td>
<td>nonce</td>
<td>nonce</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
ChaCha Stream Cipher: ChaChaN block

State Initialisation

<table>
<thead>
<tr>
<th>32b</th>
<th>32b</th>
<th>32b</th>
<th>32b</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant 0</td>
<td>constant 1</td>
<td>constant 2</td>
<td>constant 3</td>
</tr>
<tr>
<td>key 4</td>
<td>key 5</td>
<td>key 6</td>
<td>key 7</td>
</tr>
<tr>
<td>key 8</td>
<td>key 9</td>
<td>key 10</td>
<td>key 11</td>
</tr>
<tr>
<td>counter</td>
<td>nonce 13</td>
<td>nonce 14</td>
<td>nonce 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

ChaChaN Block
ChaCha Stream Cipher: ChaChaN block

Round implementation:
1. a += b; d ^= a; d <<= 16;
2. c += d; b ^= c; b <<= 12;
3. a += b; d ^= a; d <<= 8;
4. c += d; b ^= c; b <<= 7;

State Initialisation

ChaChaN Block

Ground 1  Ground 2  Ground 3  Ground 4

Ground 5  Ground 6  Ground 7  Ground 8
ChaCha Stream Cipher: ChaCha\textsubscript{N} block

State Initialisation

<table>
<thead>
<tr>
<th>32b</th>
<th>32b</th>
<th>32b</th>
<th>32b</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant 0</td>
<td>constant 1</td>
<td>constant 2</td>
<td>constant 3</td>
</tr>
<tr>
<td>key 8</td>
<td>key 9</td>
<td>key 10</td>
<td>key 11</td>
</tr>
<tr>
<td>counter 12</td>
<td>nonce 13</td>
<td>nonce 14</td>
<td>nonce 15</td>
</tr>
</tbody>
</table>

Round implementation

1. $a \leftarrow b; d \leftarrow a; d \ll 16$
2. $c \leftarrow d; b \leftarrow c; b \ll 12$
3. $a \leftarrow b; d \leftarrow a; d \ll 8$
4. $c \leftarrow d; b \leftarrow c; b \ll 7$

'N/2' times for ChaCha<\textsub-N>
ChaCha Stream Cipher: Keystream

State Initialization

512-bit keystream

Ground implementation

1. a += b; d ^= a; d <<< 16;
2. c += d; b ^= c; b <<< 12;
3. a += b; d ^= a; d <<< 8;
4. c += d; b ^= c; b <<< 7;

'N/2' times for ChaCha<N>
ChaCha Stream Cipher: Encryption/Decryption
Design Challenges
Netronome NFP-4000 Flow Processor Block Diagram
Challenge 1: Initial Nonce

Netronome NFP-4000 Flow Processor Block Diagram

Each core can process packets in parallel.
Solution 1: Use core ID as Initial Nonce

Same initial nonce → same keystream

Use processing core ID as initial nonce

<table>
<thead>
<tr>
<th></th>
<th>32b</th>
<th>32b</th>
<th>32b</th>
<th>32b</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>0</td>
<td>constant</td>
<td>1</td>
<td>constant</td>
</tr>
<tr>
<td>key</td>
<td>4</td>
<td>key</td>
<td>5</td>
<td>key</td>
</tr>
<tr>
<td>key</td>
<td>8</td>
<td>key</td>
<td>9</td>
<td>key</td>
</tr>
<tr>
<td>counter</td>
<td>12</td>
<td>nonce</td>
<td>13</td>
<td>nonce</td>
</tr>
</tbody>
</table>

Ground implementation:
1. \( a \leftarrow b \); \( d \leftarrow a \); \( d \lll 16 \);
2. \( c \leftarrow d \); \( b \leftarrow c \); \( b \lll 12 \);
3. \( a \leftarrow b \); \( d \leftarrow a \); \( d \lll 8 \);
4. \( c \leftarrow d \); \( b \leftarrow c \); \( b \lll 7 \);

512-bit keystream
Challenge 2: Nonce Generation

Next nonce?

<table>
<thead>
<tr>
<th>constant</th>
<th>key</th>
<th>key</th>
<th>key</th>
<th>key</th>
<th>nonce</th>
<th>nonce</th>
<th>nonce</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

State Initialisation

Ground implementation:
1. a ← b; d ← a; d ⋄ 16;
2. b ← c; b ⋄ 12;
3. a ← b; d ← a; d ⋄ 8;
4. c ← d; b ← c; b ⋄ 7;

'N/2' times for ChaCha<64>

512-bit keystream
Solution 2: Use previous keystream

Use 96-bit of unused keystream as next nonce
Implementation
Implementation

- Implemented on Netronome Agilio smartNIC
- Crypto primitives offered:
  - ENC - Encryption
  - DEC - Decryption
  - AUTH_set
  - AUTH_test
  - Compound primitives
    - ENC+AUTH_set
    - DEC+AUTH_test

ChaCha10

custom crc32 + ChaCha10
Implementation

Host SmartNIC in a data center network
Implementation: ENC + AUTH_set

Host SmartNIC in a data center network

- Implemented in P4
- Implemented in micro C
Implementation: ENC + AUTH_set

[Diagram of encryption and authentication process]

Host SmartNIC in a data center network

Implemented in P4
Implemented in micro C
Implementation: ENC + AUTH_set
Evaluation
Evaluation: Setup

AMD Ryzen 9 5950X (3.4 GHz, 16 cores, 32 threads) processor and 32GB RAM

Netronome Agilio CX 40 Gbit/s dual-port SmartNIC
Baseline setup

Baseline: NIC - L2 forwarding

Baseline: Host - ENC on host

AMD Ryzen 9 5950X (3.4 GHz, 16 cores, 32 threads) processor and 32GB RAM

Netronome Agilio CX 40 Gbit/s dual-port SmartNIC
Questions to answer

1. How does our implementation perform compared to the baselines?
2. Which applications will benefit by leveraging these crypto primitives?
3. How much memory is available to offload other applications?
Throughput: ChaCha based crypto primitive vs. Baseline

<table>
<thead>
<tr>
<th>Message Size</th>
<th>Improvement over Baseline:Host</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>64 B</td>
<td>46x</td>
</tr>
<tr>
<td>128B</td>
<td>33x</td>
</tr>
<tr>
<td>256B</td>
<td>23x</td>
</tr>
</tbody>
</table>

Meets application requirements:
- Serverless: 20K to 100K pps
- Host-INT: 3.3M pps (18B pkt)
Latency: ChaCha based crypto primitive vs. Baseline

<table>
<thead>
<tr>
<th>Message Size</th>
<th>Latency reduction over Baseline:Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>Min: 69%</td>
</tr>
<tr>
<td>128B</td>
<td>Min: 70%</td>
</tr>
<tr>
<td>256B</td>
<td>Min: 59%</td>
</tr>
</tbody>
</table>
Summary

- Implemented in-network ChaCha crypto without using co-processor
- Solution meets crypto processing requirements of control applications

Future Work

- Implementing Poly-1305 authentication algorithm
- Crypto processing for MTU-sized messages
- Crypto primitive APIs for P4/C programmers

Contact: shaguftha21079@iiitd.ac.in