NEON crypto

Daniel J. Bernstein, Peter Schwabe

September 11, 2012

CHES 2012, Leuven, Belgium

NEON

- Target of this paper: Make cryptography fast on a large class of mobile devices, e.g.,

  Apple iPhone 3GS, Apple iPhone 4, 3rd generation Apple iPod touch (late 2009), Apple iPad 1, Nokia N9, Nokia N900, Palm Pre Plus, Samsung/Google Nexus S, Samsung Galaxy S, ...

NEON

- Target of this paper: Make cryptography fast on a large class of mobile devices, e.g.,

  Apple iPhone 3GS, Apple iPhone 4, 3rd generation Apple iPod touch (late 2009), Apple iPad 1, Nokia N9, Nokia N900, Palm Pre Plus, Samsung/Google Nexus S, Samsung Galaxy S, ...
- All these devices have an ARM Cortex-A8 CPU with NEON vector instruction set

NEON

- Target of this paper: Make cryptography fast on a large class of mobile devices, e.g.,

  Apple iPhone 3GS, Apple iPhone 4, 3rd generation Apple iPod touch (late 2009), Apple iPad 1, Nokia N9, Nokia N900, Palm Pre Plus, Samsung/Google Nexus S, Samsung Galaxy S, ...
- All these devices have an ARM Cortex-A8 CPU with NEON vector instruction set
- Many more devices with NEON:

  HTC Sensation, HTC 7 Mozart, HTC Desire, HTC/Google Nexus One, LG Optimus 7, Motorola Droid Bionic, Nokia Lumia, Samsung/Google Galaxy Nexus, Samsung Galaxy S II and S III, ...

NEON

- Target of this paper: Make cryptography fast on a large class of mobile devices, e.g.,

  Apple iPhone 3GS, Apple iPhone 4, 3rd generation Apple iPod touch (late 2009), Apple iPad 1, Nokia N9, Nokia N900, Palm Pre Plus, Samsung/Google Nexus S, Samsung Galaxy S, ...
- All these devices have an ARM Cortex-A8 CPU with NEON vector instruction set
- Many more devices with NEON:

  HTC Sensation, HTC 7 Mozart, HTC Desire, HTC/Google Nexus One, LG Optimus 7, Motorola Droid Bionic, Nokia Lumia, Samsung/Google Galaxy Nexus, Samsung Galaxy S II and S III, ...
- Those devices have Cortex-A9 and Qualcomm Snapdragon CPUs

NEON

- Target of this paper: Make cryptography fast on a large class of mobile devices, e.g.,

  Apple iPhone 3GS, Apple iPhone 4, 3rd generation Apple iPod touch (late 2009), Apple iPad 1, Nokia N9, Nokia N900, Palm Pre Plus, Samsung/Google Nexus S, Samsung Galaxy S, ...
- All these devices have an ARM Cortex-A8 CPU with NEON vector instruction set
- Many more devices with NEON:

  HTC Sensation, HTC 7 Mozart, HTC Desire, HTC/Google Nexus One, LG Optimus 7, Motorola Droid Bionic, Nokia Lumia, Samsung/Google Galaxy Nexus, Samsung Galaxy S II and S III, ...
- Those devices have Cortex-A9 and Qualcomm Snapdragon CPUs
- Rest of this talk: Focus on NEON in Cortex-A8

crypto

- Obvious target algorithm: AES with 128-bit key

crypto

- Obvious target algorithm: AES with 128-bit key
- Best previous result: Krovetz and Rogaway report 25.4 cycles/byte for implementation by Polyakov, included in OpenSSL
- Not protected against timing attacks

crypto

- Obvious target algorithm: AES with 128-bit key
- Best previous result: Krovetz and Rogaway report 25.4 cycles/byte for implementation by Polyakov, included in OpenSSL
- Not protected against timing attacks
- For constant-time implementation: Bitsliced approach by Käsper and Schwabe (CHES 2009), logical operations on 8 blocks in parallel

crypto

- Obvious target algorithm: AES with 128-bit key
- Best previous result: Krovetz and Rogaway report 25.4 cycles/byte for implementation by Polyakov, included in OpenSSL
- Not protected against timing attacks
- For constant-time implementation: Bitsliced approach by Käsper and Schwabe (CHES 2009), logical operations on 8 blocks in parallel
- Per round of AES: 167 logical operations (148 in the last round)
- Total of $9 \cdot (167) + 148 = 1651$ logical operations

crypto

- Obvious target algorithm: AES with 128-bit key
- Best previous result: Krovetz and Rogaway report 25.4 cycles/byte for implementation by Polyakov, included in OpenSSL
- Not protected against timing attacks
- For constant-time implementation: Bitsliced approach by Käsper and Schwabe (CHES 2009), logical operations on 8 blocks in parallel
- Per round of AES: 167 logical operations (148 in the last round)
- Total of $9 \cdot (167) + 148 = 1651$ logical operations
- NEON can do one logical operation per cycle
- Lower bound of 1651 cycles/8 blocks; 12.898 cycles/byte

crypto

- Obvious target algorithm: AES with 128-bit key
- Best previous result: Krovetz and Rogaway report 25.4 cycles/byte for implementation by Polyakov, included in OpenSSL
- Not protected against timing attacks
- For constant-time implementation: Bitsliced approach by Käsper and Schwabe (CHES 2009), logical operations on 8 blocks in parallel
- Per round of AES: 167 logical operations (148 in the last round)
- Total of $9 \cdot (167) + 148 = 1651$ logical operations
- NEON can do one logical operation per cycle
- Lower bound of 1651 cycles/8 blocks; 12.898 cycles/byte
- This ignores cost for bitslice transformation, xoring of keystream in CTR mode ...
- Our AES NEON speed: 18.94 cycles/byte, constant time

crypto: there's more!

- Cryptographic primitives required for secure network communication:
  - Symmetric encryption
  - Secret-key authentication
  - Key exchange (Diffie-Hellman)
  - Public-key signatures

crypto: there's more!

- Cryptographic primitives required for secure network communication:
  - Symmetric encryption
  - Secret-key authentication
  - Key exchange (Diffie-Hellman)
  - Public-key signatures
- At least 128 bits of security

crypto: there's more!

- Cryptographic primitives required for secure network communication:
  - Symmetric encryption
  - Secret-key authentication
  - Key exchange (Diffie-Hellman)
  - Public-key signatures
- At least 128 bits of security
- Protection against timing attacks

crypto: there's more!

- Cryptographic primitives required for secure network communication:
  - Symmetric encryption
  - Secret-key authentication
  - Key exchange (Diffie-Hellman)
  - Public-key signatures
- At least 128 bits of security
- Protection against timing attacks
- As fast as possible on ARM Cortex-A8

crypto: there's more!

- Cryptographic primitives required for secure network communication:
  - Symmetric encryption
  - Secret-key authentication
  - Key exchange (Diffie-Hellman)
  - Public-key signatures
- At least 128 bits of security
- Protection against timing attacks
- As fast as possible on ARM Cortex-A8
- Our choice of primitives:
  - Salsa20
  - Poly1305
  - Curve25519
  - Ed25519

Salsa20

- Designed by Bernstein in 2005; recommended in the eSTREAM software portfolio
- Generates random stream in 64-byte blocks, works on 32-bit integers
- Per block: 20 rounds; each round doing 16 add-rotate-xor sequences, such as

```
s4 = x0 + x12
x4 ^= (s4 >>> 25)
```

Salsa20

- Designed by Bernstein in 2005; recommended in the eSTREAM software portfolio
- Generates random stream in 64-byte blocks, works on 32-bit integers
- Per block: 20 rounds; each round doing 16 add-rotate-xor sequences, such as

```
s4 = x0 + x12
x4 ^= (s4 >>> 25)
```

- In ARM *without* NEON: 2 instructions, 1 cycle
- Sounds like total of $(20 \cdot 16)/64 = 5$ cycles/byte

Salsa20

- Designed by Bernstein in 2005; recommended in the eSTREAM software portfolio
- Generates random stream in 64-byte blocks, works on 32-bit integers
- Per block: 20 rounds; each round doing 16 add-rotate-xor sequences, such as

```
s4 = x0 + x12
x4 ^= (s4 >>> 25)
```

- In ARM *without* NEON: 2 instructions, 1 cycle
- Sounds like total of $(20 \cdot 16)/64 = 5$ cycles/byte, but:
  - Only 14 integer registers (need at least 17)
  - Latencies cause big trouble
  - Actual implementations were slower than 15 cycles/byte

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

- This has 9 cycles latency: Need at least $(9 \cdot 20 \cdot 4)/64 = 11.25$ cycles/byte

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

- This has 9 cycles latency: Need at least $(9 \cdot 20 \cdot 4)/64 = 11.25$ cycles/byte
- Blocks are independent, interleave two blocks; need at least 6.875 cycles/byte

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

- This has 9 cycles latency: Need at least $(9 \cdot 20 \cdot 4)/64 = 11.25$ cycles/byte
- Blocks are independent, interleave two blocks; need at least 6.875 cycles/byte
- ... interleave three blocks; need at least 6.25 cycles/byte

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

- This has 9 cycles latency: Need at least $(9 \cdot 20 \cdot 4)/64 = 11.25$ cycles/byte
- Blocks are independent, interleave two blocks; need at least 6.875 cycles/byte
- ... interleave three blocks; need at least 6.25 cycles/byte
- The ARM unit is still idle, so interleave ARM with NEON:
  - One block on ARM, two blocks on NEON
  - Bottleneck: decode at most 2 instructions per cycle

Salsa20 on the Cortex-A8

- Add-rotate-xor sequences are 4-way parallel, good for SIMD
- Rotates are not free, cost 3 instructions:

```
4x a0 = diag1 + diag0
4x b0 = a0 << 7
4x a0 unsigned>>= 25
   diag3 ^= b0
   diag3 ^= a0
```

- This has 9 cycles latency: Need at least $(9 \cdot 20 \cdot 4)/64 = 11.25$ cycles/byte
- Blocks are independent, interleave two blocks; need at least 6.875 cycles/byte
- ... interleave three blocks; need at least 6.25 cycles/byte
- The ARM unit is still idle, so interleave ARM with NEON:
  - One block on ARM, two blocks on NEON
  - Bottleneck: decode at most 2 instructions per cycle
- Final result, including overhead: 5.47 cycles/byte

Poly1305

- Designed by Bernstein in 2005
- Secret-key one-time authenticator based on arithmetic in $\mathbb{F}_p$ with $p = 2^{130} - 5$
- Key k and (padded) 16-byte ciphertext blocks $c_1, \ldots, c_k$ are in $\mathbb{F}_p$

Poly1305

- Designed by Bernstein in 2005
- Secret-key one-time authenticator based on arithmetic in $\mathbb{F}_p$ with $p = 2^{130} - 5$
- Key k and (padded) 16-byte ciphertext blocks $c_1, \ldots, c_k$ are in $\mathbb{F}_p$
- Main work: initialize authentication tag h with 0, then compute:

  for i from 1 to k **do**
  $h \leftarrow h + c_i$
  $h \leftarrow h \cdot k$
  end for

Poly1305

- Designed by Bernstein in 2005
- Secret-key one-time authenticator based on arithmetic in $\mathbb{F}_p$ with $p = 2^{130} - 5$
- Key k and (padded) 16-byte ciphertext blocks $c_1, \ldots, c_k$ are in $\mathbb{F}_p$
- Main work: initialize authentication tag h with 0, then compute:

  for i from 1 to k **do**
  $h \leftarrow h + c_i$
  $h \leftarrow h \cdot k$
  end for

- Per 16 bytes: 1 multiplication, 1 addition in $\mathbb{F}_{2^{130}-5}$
- Some (fast) finalization to produce 16-byte authentication tag

Poly1305

- Designed by Bernstein in 2005
- Secret-key one-time authenticator based on arithmetic in $\mathbb{F}_p$ with $p = 2^{130} - 5$
- Key k and (padded) 16-byte ciphertext blocks $c_1, \ldots, c_k$ are in $\mathbb{F}_p$
- Main work: initialize authentication tag h with 0, then compute:

  for i from 1 to k **do**
  $h \leftarrow h + c_i$
  $h \leftarrow h \cdot k$
  end for

- Per 16 bytes: 1 multiplication, 1 addition in $\mathbb{F}_{2^{130}-5}$
- Some (fast) finalization to produce 16-byte authentication tag

Poly1305 on the Cortex-A8

- Fastest NEON multiplier: Two SIMD $32 \times 32 \rightarrow 64$ bit integer multiplications every two cycles
- Multiply-accumulate at the same cost as multiply

Poly1305 on the Cortex-A8

- Fastest NEON multiplier: Two SIMD $32 \times 32 \to 64$ bit integer multiplications every two cycles
- Multiply-accumulate at the same cost as multiply
- NEON additions lose carry bits; we need a carry-safe (redundant) representation
- Represent an element A of $\mathbb{F}_p$ as $(a_0, a_1, a_2, a_3, a_4)$ with

$$A = \sum_{i=0}^{4} a_i \cdot 2^{26 \cdot i}$$

Poly1305 on the Cortex-A8

- Fastest NEON multiplier: Two SIMD $32 \times 32 \to 64$ bit integer multiplications every two cycles
- Multiply-accumulate at the same cost as multiply
- NEON additions lose carry bits; we need a carry-safe (redundant) representation
- Represent an element A of $\mathbb{F}_p$ as $(a_0, a_1, a_2, a_3, a_4)$ with

$$A = \sum_{i=0}^{4} a_i \cdot 2^{26 \cdot i}$$

- In multiplication of $C = A \cdot B$ obtain coefficients $c_0, c_1, \ldots, c_8$
- Reduction: $2^{130} \equiv 5 \pmod{p}$. Hence add $5c_5$ to c_0, $5c_6$ to c_1, etc.

Poly1305 on the Cortex-A8

- Schoolbook multiplication breaks into 25 32-bit integer multiplications and 16 64-bit additions
- Many of those are parallel, can perform them in SIMD

Poly1305 on the Cortex-A8

- Schoolbook multiplication breaks into 25 32-bit integer multiplications and 16 64-bit additions
- Many of those are parallel, can perform them in SIMD, but
  - this requires quite a bit of shuffling, and
  - latencies in the final carry chain kick in

Poly1305 on the Cortex-A8

- Schoolbook multiplication breaks into 25 32-bit integer multiplications and 16 64-bit additions
- Many of those are parallel, can perform them in SIMD, but
  - this requires quite a bit of shuffling, and
  - latencies in the final carry chain kick in
- Better: Precompute k^2
- Compute $((c_0 \cdot k) + c_1) \cdot k$ as $(c_0 \cdot k^2) + (c_1 \cdot k)$

Poly1305 on the Cortex-A8

- Schoolbook multiplication breaks into 25 32-bit integer multiplications and 16 64-bit additions
- Many of those are parallel, can perform them in SIMD, but
  - this requires quite a bit of shuffling, and
  - latencies in the final carry chain kick in
- Better: Precompute k^2
- Compute $((c_0 \cdot k) + c_1) \cdot k$ as $(c_0 \cdot k^2) + (c_1 \cdot k)$
- Always perform two independent multiplications in $\mathbb{F}_p$ together in SIMD

Poly1305 on the Cortex-A8

- Schoolbook multiplication breaks into 25 32-bit integer multiplications and 16 64-bit additions
- Many of those are parallel, can perform them in SIMD, but
  - this requires quite a bit of shuffling, and
  - latencies in the final carry chain kick in
- Better: Precompute k^2
- Compute $((c_0 \cdot k) + c_1) \cdot k$ as $(c_0 \cdot k^2) + (c_1 \cdot k)$
- Always perform two independent multiplications in $\mathbb{F}_p$ together in SIMD
- Final result: 2.20 cycles/byte

Curve25519 and Ed25519

- Curve25519: ECDH key exchange (Bernstein, PKC 2006)
- Ed25519: Elliptic-curve signatures (Bernstein, Duif, Lange, Schwabe, Yang, CHES 2011)

Curve25519 and Ed25519

- Curve25519: ECDH key exchange (Bernstein, PKC 2006)
- Ed25519: Elliptic-curve signatures (Bernstein, Duif, Lange, Schwabe, Yang, CHES 2011)
- Arithmetic on Montgomery curve or birationally equivalent twisted Edwards curve over $\mathbb{F}_{2^{255}-19}$
- Again, use redundant representation: $A = (a_0, \ldots, a_9)$, with

$$A = \sum_{i=0}^{9} a_i \cdot 2^{\lceil 25.5 \cdot i \rceil}$$

Curve25519 and Ed25519

- Curve25519: ECDH key exchange (Bernstein, PKC 2006)
- Ed25519: Elliptic-curve signatures (Bernstein, Duif, Lange, Schwabe, Yang, CHES 2011)
- Arithmetic on Montgomery curve or birationally equivalent twisted Edwards curve over $\mathbb{F}_{2^{255}-19}$
- Again, use redundant representation: $A = (a_0, \ldots, a_9)$, with

$$A = \sum_{i=0}^{9} a_i \cdot 2^{\lceil 25.5 \cdot i \rceil}$$

- Similar ideas to Poly1305:
  - Efficient reduction through $2^{255} \equiv 19$: add $19c_{10}$ to c_0, etc.
  - Whenever possible, perform two independent multiplications or squarings together

Curve25519 and Ed25519

- Curve25519: ECDH key exchange (Bernstein, PKC 2006)
- Ed25519: Elliptic-curve signatures (Bernstein, Duif, Lange, Schwabe, Yang, CHES 2011)
- Arithmetic on Montgomery curve or birationally equivalent twisted Edwards curve over $\mathbb{F}_{2^{255}-19}$
- Again, use redundant representation: $A = (a_0, \ldots, a_9)$, with

$$A = \sum_{i=0}^{9} a_i \cdot 2^{\lceil 25.5 \cdot i \rceil}$$

- Similar ideas to Poly1305:
  - Efficient reduction through $2^{255} \equiv 19$: add $19c_{10}$ to c_0, etc.
  - Whenever possible, perform two independent multiplications or squarings together
- Constant-time conditional swaps (Curve25519) and table lookups (Ed25519) to protect against timing attacks

Results & Outlook

- Secret-key authenticated encryption: 7.67 cycles/byte, > 830 MBit/sec on 800 MHz Cortex-A8
  - Salsa20: 5.47 cycles/byte
  - Poly1305: 2.20 cycles/byte
- Compute shared secret (ECDH): 492417 cycles (> 1600/sec)
- All software is timing-attack resistant

Results & Outlook

- Secret-key authenticated encryption: 7.67 cycles/byte, > 830 MBit/sec on 800 MHz Cortex-A8
  - Salsa20: 5.47 cycles/byte
  - Poly1305: 2.20 cycles/byte
- Compute shared secret (ECDH): 492417 cycles (> 1600/sec)
- All software is timing-attack resistant
- Also Cortex-A9 and Qualcomm Snapdragon CPUs benefit from the software speedups
- Still required: Microarchitecture-specific optimization for those

Results & Outlook

- Secret-key authenticated encryption: 7.67 cycles/byte, > 830 MBit/sec on 800 MHz Cortex-A8
  - Salsa20: 5.47 cycles/byte
  - Poly1305: 2.20 cycles/byte
- Compute shared secret (ECDH): 492417 cycles (> 1600/sec)
- All software is timing-attack resistant
- Also Cortex-A9 and Qualcomm Snapdragon CPUs benefit from the software speedups
- Still required: Microarchitecture-specific optimization for those
- Followup result by Hamburg:
  - Use similar ECC techniques, slightly smaller curve
  - Use more powerful ARM core on Cortex-A9
  - Don't use NEON
  - Compute shared secret (ECDH): 616000 cycles
- Obvious question: How far can we go on Cortex-A9 *with* NEON?

Results & Outlook

- Secret-key authenticated encryption: 7.67 cycles/byte, > 830 MBit/sec on 800 MHz Cortex-A8
  - Salsa20: 5.47 cycles/byte
  - Poly1305: 2.20 cycles/byte
- Compute shared secret (ECDH): 492417 cycles (> 1600/sec)
- All software is timing-attack resistant
- Also Cortex-A9 and Qualcomm Snapdragon CPUs benefit from the software speedups
- Still required: Microarchitecture-specific optimization for those
- Followup result by Hamburg:
  - Use similar ECC techniques, slightly smaller curve
  - Use more powerful ARM core on Cortex-A9
  - Don't use NEON
  - Compute shared secret (ECDH): 616000 cycles
- Obvious question: How far can we go on Cortex-A9 *with* NEON?
- Future: target low-power energy-efficient Cortex-A7

NEON crypto online

- The paper is online at http://cryptojedi.org/papers/#neoncrypto
- NEON AES-128-CTR, Salsa20, Poly1305 now in SUPERCOP: http://bench.cr.yp.to
- We're still speeding up Curve25519, Ed25519 but will include them in SUPERCOP
- All software in the public domain
- Software to be included in the next release of the NaCl library: http://nacl.cr.yp.to