

NaCl on 8-bit AVR microcontrollers

Michael Hutter and Peter Schwabe

TU Graz (Austria) and Radboud University Nijmegen (The Netherlands)



June 24, 2013

Africacrypt 2013, Cairo, Egypt

... almost 2 years ago in Nara, Japan



The plan

- ▶ Bring Ed25519 elliptic-curve signatures to 8-bit AVR microcontroller

The plan

- ▶ Bring Ed25519 elliptic-curve signatures to 8-bit AVR microcontroller
- ▶ Write paper, submit to Africacrypt 2012

The plan

- ▶ Bring Ed25519 elliptic-curve signatures to 8-bit AVR microcontroller
- ▶ Write paper, submit to Africacrypt 2012
- ▶ Hopefully get accepted, go to Morocco

The plan

- ▶ Bring Ed25519 elliptic-curve signatures to 8-bit AVR microcontroller
- ▶ Write paper, submit to Africacrypt 2012
- ▶ Hopefully get accepted, go to Morocco

... what happened?

- ▶ Update the plan: Get the whole Networking and Cryptography Library (NaCl) onto AVR
- ▶ Write paper about it, submit to Africacrypt 2013
- ▶ Get accepted, go to Egypt

8-bit AVR microcontrollers

- ▶ Widely used in embedded systems, e.g., sensor nodes
- ▶ 3 product lines: ATxmega, ATmega, and ATtiny (no HW multiplier)
- ▶ Focus here: ATmega, example configurations:
 - ▶ **ATmega2560**: 16 MHz, 256 KB flash, 8 KB RAM
 - ▶ **ATmega128**: 16 MHz, 128 KB flash, 4 KB RAM
 - ▶ **ATmega328**: 20 MHz, 32 KB flash, 2 KB RAM

8-bit AVR microcontrollers

- ▶ Widely used in embedded systems, e.g., sensor nodes
- ▶ 3 product lines: ATxmega, ATmega, and ATtiny (no HW multiplier)
- ▶ Focus here: ATmega, example configurations:
 - ▶ **ATmega2560**: 16 MHz, 256 KB flash, 8 KB RAM
 - ▶ **ATmega128**: 16 MHz, 128 KB flash, 4 KB RAM
 - ▶ **ATmega328**: 20 MHz, 32 KB flash, 2 KB RAM
- ▶ RISC architecture (> 90 available instructions)
- ▶ 32 general purpose registers
 - ▶ R1:R0 holds 16-bit multiplication result
 - ▶ R16-R31 accessible by a limited set of instructions
 - ▶ R26-R31 (X, Y, and Z) used for 16-bit addressing

8-bit AVR microcontrollers

- ▶ Widely used in embedded systems, e.g., sensor nodes
- ▶ 3 product lines: ATxmega, ATmega, and ATtiny (no HW multiplier)
- ▶ Focus here: ATmega, example configurations:
 - ▶ **ATmega2560**: 16 MHz, 256 KB flash, 8 KB RAM
 - ▶ **ATmega128**: 16 MHz, 128 KB flash, 4 KB RAM
 - ▶ **ATmega328**: 20 MHz, 32 KB flash, 2 KB RAM
- ▶ RISC architecture (> 90 available instructions)
- ▶ 32 general purpose registers
 - ▶ R1-R0 holds 16-bit multiplication result
 - ▶ R16-R31 accessible by a limited set of instructions
 - ▶ R26-R31 (X, Y, and Z) used for 16-bit addressing
- ▶ We performed benchmarks on the ATmega2560

NaCl: A new cryptographic library

- ▶ Networking and Cryptography library (NaCl, pronounced “salt”)
- ▶ Designed by Daniel J. Bernstein, Tanja Lange, Peter Schwabe
- ▶ Acknowledgment: Contributions by
 - ▶ Matthew Dempsky (Mochi Media)
 - ▶ Niels Duif (TU Eindhoven)
 - ▶ Emilia Käsper (KU Leuven, now Google)
 - ▶ Adam Langley (Google)
 - ▶ Bo-Yin Yang (Academia Sinica)
- ▶ Public domain, no patents
- ▶ Used, for example, in OpenDNS, DNSCrypt, QuickTun VPN, and Ethos OS

NaCl features

- ▶ Easy-to-use API:
 - ▶ One function call to `crypto_box` to generate public-key authenticated ciphertext
 - ▶ One function call to `crypto_sign` to sign a message
 - ▶ No error handling required, no memory allocation required

NaCl features

- ▶ Easy-to-use API:
 - ▶ One function call to `crypto_box` to generate public-key authenticated ciphertext
 - ▶ One function call to `crypto_sign` to sign a message
 - ▶ No error handling required, no memory allocation required
- ▶ Only \geq 128-bit-secure cryptographic primitives

NaCl features

- ▶ Easy-to-use API:
 - ▶ One function call to `crypto_box` to generate public-key authenticated ciphertext
 - ▶ One function call to `crypto_sign` to sign a message
 - ▶ No error handling required, no memory allocation required
- ▶ Only \geq 128-bit-secure cryptographic primitives
- ▶ Timing-attack protection:
 - ▶ No load/store addresses that depend on secret data (no cache timing!)
 - ▶ No branch conditions that depend on secret data

NaCl features

- ▶ Easy-to-use API:
 - ▶ One function call to `crypto_box` to generate public-key authenticated ciphertext
 - ▶ One function call to `crypto_sign` to sign a message
 - ▶ No error handling required, no memory allocation required
- ▶ Only \geq 128-bit-secure cryptographic primitives
- ▶ Timing-attack protection:
 - ▶ No load/store addresses that depend on secret data (no cache timing!)
 - ▶ No branch conditions that depend on secret data
- ▶ Very high speed

NaCl features

- ▶ Easy-to-use API:
 - ▶ One function call to `crypto_box` to generate public-key authenticated ciphertext
 - ▶ One function call to `crypto_sign` to sign a message
 - ▶ No error handling required, no memory allocation required
- ▶ Only \geq 128-bit-secure cryptographic primitives
- ▶ Timing-attack protection:
 - ▶ No load/store addresses that depend on secret data (no cache timing!)
 - ▶ No branch conditions that depend on secret data
- ▶ Very high speed ... on large desktop/server processors

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs
- ▶ Optimize algorithms *across* primitives to reuse more code

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs
- ▶ Optimize algorithms *across* primitives to reuse more code
- ▶ Memory access is uncached: secret load addresses are not a problem!

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs
- ▶ Optimize algorithms *across* primitives to reuse more code
- ▶ Memory access is uncached: secret load addresses are not a problem!
- ▶ No branch prediction, but still: avoid secret branch conditions
 - ▶ Different cost for branch instructions on different AVR
 - ▶ Much easier to check than constant-time branches

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs
- ▶ Optimize algorithms *across* primitives to reuse more code
- ▶ Memory access is uncached: secret load addresses are not a problem!
- ▶ No branch prediction, but still: avoid secret branch conditions
 - ▶ Different cost for branch instructions on different AVRs
 - ▶ Much easier to check than constant-time branches
- ▶ So far: No secure randomness generation (compute keys outside)

NaCl on ATmegs

- ▶ Target: Provide reasonable size-speed tradeoffs
- ▶ Optimize algorithms *across* primitives to reuse more code
- ▶ Memory access is uncached: secret load addresses are not a problem!
- ▶ No branch prediction, but still: avoid secret branch conditions
 - ▶ Different cost for branch instructions on different AVR
 - ▶ Much easier to check than constant-time branches
- ▶ So far: No secure randomness generation (compute keys outside)
- ▶ Addresses have only 16 bits, so restrict message length to $2^{16} - 1$ (avoid expensive arithmetic on 64-bit integers)

Under the hood of NaCl

Under the hood of `crypto_box`

- ▶ Curve25519 elliptic-curve Diffie-Hellman
- ▶ Subsequent secret-key authenticated encryption
- ▶ Stream cipher: Salsa20
- ▶ Authenticator: Poly1305
- ▶ Note: allows repudiation

Under the hood of `crypto_sign`

- ▶ Ed25519 elliptic-curve signatures
- ▶ Support for fast batch verification

Salsa20

- ▶ Stream cipher proposed in 2005 (within the eSTREAM project)
- ▶ Consists of 20 rounds and a 64-byte state (4×4 32-bit state)
- ▶ We implemented two API entry points in C
 - ▶ `crypto_stream`: generates a pseudorandom stream
 - ▶ `crypto_stream_xor`: XORs the stream with a message and outputs the ciphertext
- ▶ Core functionality (`crypto_core`) implemented in assembly (initialization and round calculations)
- ▶ 80 quarterround function calls on either a row or a column

Salsa20 optimization

- ▶ Parameter passing using registers (no costly stack usage)
- ▶ Content of the state is modified *in-place* (no variables, copies, etc.)
- ▶ Shifts by 7 and 9 are cheap logical shifts (LSR and LSL)
- ▶ Shifts by 13 and 18 have been realized using MUL
- ▶ 176 cycles for one quarterround function call

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, \dots, 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 optimization

- ▶ 17×17 -byte multiplication
 - ▶ Split 136-bit multiplication into 8×8 , 9×9 , and 9×8 -byte multiplications
 - ▶ Partial products are processed according to schoolbook multiplication
 - ▶ Performance: 1,882 cycles and 2,944 bytes of code (unrolled)
- ▶ Reduction mod $2^{130} - 5$
 - ▶ We applied fast reduction by exploiting the congruence $2^{130} \equiv 5$
 - ▶ Can be done by cheap shifts and additions on AVRs
 - ▶ Re-use of `bigint_add` which is also used for scalar arithmetic in Ed25519

Curve25519 and Ed25519

Curve25519

- ▶ Elliptic-curve Diffie-Hellman protocol proposed by Bernstein in 2006
- ▶ Uses Montgomery curve over the field $\mathbb{F}_{2^{255}-19}$
- ▶ Main operation: 253-step Montgomery ladder using $(X : Z)$ -coordinates

Ed25519

- ▶ Elliptic-curve signatures proposed by Bernstein, Duif, Lange, Schwabe, and Yang in 2011
- ▶ Based on Schnorr signatures with some modifications
- ▶ Use twisted Edwards curve birationally equivalent to Curve25519
- ▶ Signing: fixed-base-point scalar multiplication
- ▶ Verification: point decompression + double-scalar multiplication
- ▶ Uses SHA-512 as hash function (plan: update to SHA-3)

*25519 optimization

- ▶ Implemented Karatsuba's technique
- ▶ 256-bit multiplication is split into two 16×16 and one 17×17 multiplication
- ▶ Allows us to re-use code of Poly1305
- ▶ For Ed25519, we stored pre-computed multiples of the base point in flash memory and used a window size of 4 (high speed) and 2 (low area)
- ▶ SHA-512: 64-bit transformations have been implemented in assembly

AVR NaCl results

High-speed configuration

- ▶ Secret-key authenticated encryption: ≈ 500 cycles/byte (268 bytes of RAM)
- ▶ Variable-basepoint scalar multiplication: 22,791,580 cycles (677 bytes of RAM)
- ▶ `crypto_sign`: 23,216,241 cycles (1,642 bytes of RAM)
- ▶ `crypto_sign_open`: 32,634,713 cycles (1,315 bytes of RAM)
- ▶ 27,962 bytes of ROM for NaCl

AVR NaCl results

Small-size configuration

- ▶ Secret-key authenticated encryption: ≈ 520 cycles/byte (273 bytes of RAM)
- ▶ Variable-basepoint scalar multiplication: 27,926,288 cycles (917 bytes of RAM)
- ▶ `crypto_sign`: 34,303,972 cycles (1,289 bytes of RAM)
- ▶ `crypto_sign_open`: 40,083,281 cycles (1,346 bytes of RAM)
- ▶ 17,373 bytes of ROM for NaCl

Summary

- ▶ First implementation of NaCl on AVR
- ▶ New speed records for Salsa20 on AVR
- ▶ First Poly1305, Curve25519, and Ed25519 results on AVR
- ▶ Fully compatible framework to other (already existing) NaCl implementations for, e.g., servers, laptops, mobile phones,
- ▶ 128-bit security level
- ▶ Full protection against timing attacks

Future work (things we don't have, yet)

- ▶ Core algorithms are implemented, *not* the whole API, yet (in particular no `crypto_box`, yet)
- ▶ No flexible build system, yet
- ▶ Need more tradoffs, in particular for even smaller size
- ▶ Further optimizations in assembly
- ▶ Investigate protection against physical side-channel attacks

More NaCl online

- ▶ NaCl website: <http://nacl.cr.yp.to>
- ▶ This paper: <http://cryptojedi.org/papers/#avrnacl>
- ▶ Software: <http://cryptojedi.org/crypto/#avrnacl>