NaCl (for the IoT)

Peter Schwabe

Radboud University, Nijmegen, The Netherlands

November 18, 2015

Sensemakers IoT Meeting, Amsterdam

A bit about me

2001-2006: Studies of computer science at RWTH Aachen

A bit about me

2008-2011: Ph.D. student at TU Eindhoven

A bit about me

2011-2012: Postdoc at Academia Sinica and NTU (Taiwan)

A bit about me

Since Jan. 2013: Assistant professor at Radboud University

Space shuttles and elevators

"OpenSSL is the space shuttle of crypto libraries. It will get you to space, provided you have a team of people to push the ten thousand buttons required to do so. NaCl is more like an elevator – you just press a button and it takes you there. No frills or options.

I like elevators."

Matthew Green in his blog entry *The anatomy of a bad idea*

NaCl: Networking and Cryptography library

- Core development team: 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)

NaCl: Networking and Cryptography library

- Core development team: 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)
- User's perspective: Bundle of functionalities rather than bundle of algorithms
- Focus on protecting Internet communication

Protecting Internet communication

- Alice wants to send a message m to Bob
- Uses Bob's public key and her own private key to compute authenticated ciphertext c, sends c to Bob
- Bob uses his private key and Alice's public key to verify and recover m

Alice using a typical crypto library

- First choose algorithms and parameters, e.g. AES-128, RSA-2048, SHA-256
- Generate random AES key
- Use AES to encrypt packet
- Hash encrypted packet
- Read RSA private key from wire format
- Use key to sign hash
- Read Bob's RSA public key from wire format
- Use key to encrypt AES key and signature
- ...

Alice using a typical crypto library

- First choose algorithms and parameters, e.g. AES-128, RSA-2048, SHA-256
- Generate random AES key
- Use AES to encrypt packet
- Hash encrypted packet
- Read RSA private key from wire format
- Use key to sign hash
- Read Bob's RSA public key from wire format
- Use key to encrypt AES key and signature
- ...
- Plus more code to allocate storage, handle errors etc.

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

- `sk`: Alice's 32-byte private key
- `pk`: Bob's 32-byte public key
- `n`: 24-byte nonce
- `c`: authenticated ciphertext, 16 bytes longer than plaintext `m`

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

- `sk`: Alice's 32-byte private key
- `pk`: Bob's 32-byte public key
- `n`: 24-byte nonce
- `c`: authenticated ciphertext, 16 bytes longer than plaintext `m`
- All objects are C++ `std::string` variables represented in wire format, ready for transmission

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

- `sk`: Alice's 32-byte private key
- `pk`: Bob's 32-byte public key
- `n`: 24-byte nonce
- `c`: authenticated ciphertext, 16 bytes longer than plaintext `m`
- All objects are C++ `std::string` variables represented in wire format, ready for transmission
- C NaCl is similar; using pointers, no memory allocation, no errors

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

- `sk`: Alice's 32-byte private key
- `pk`: Bob's 32-byte public key
- `n`: 24-byte nonce
- `c`: authenticated ciphertext, 16 bytes longer than plaintext `m`
- All objects are C++ `std::string` variables represented in wire format, ready for transmission
- C NaCl is similar; using pointers, no memory allocation, no errors
- Bob verifies and decrypts:

```
m = crypto_box_open(c,n,pk,sk)
```

Alice using NaCl

```
c = crypto_box(m,n,pk,sk)
```

- `sk`: Alice's 32-byte private key
- `pk`: Bob's 32-byte public key
- `n`: 24-byte nonce
- `c`: authenticated ciphertext, 16 bytes longer than plaintext `m`
- All objects are C++ `std::string` variables represented in wire format, ready for transmission
- C NaCl is similar; using pointers, no memory allocation, no errors
- Bob verifies and decrypts:

```
m = crypto_box_open(c,n,pk,sk)
```

- Initial keypair generation for Alice and Bob:

```
pk = crypto_box_keypair(&sk)
```

Signatures in NaCl

- `crypto_box` does not use signatures but a public-key authenticator
- Sometimes non-repudiability is required or one wants broadcast authenticated communication

Signatures in NaCl

- `crypto_box` does not use signatures but a public-key authenticator
- Sometimes non-repudiability is required or one wants broadcast authenticated communication
- NaCl also contains signatures with an easy-to-use interface:

  ```
  pk = crypto_sign_keypair(&sk)
  ```

  generates a 64-byte private key and a 32-byte public key

  ```
  sm = crypto_sign(m, sk)
  ```

  signs `m` under `sk`; `sm` is 64 bytes longer than `m`

  ```
  m = crypto_sign_open(sm, pk)
  ```

  verifies the signature and recovers `m`

NaCl Security: No secret load addresses

- Osvik, Shamir, and Tromer in 2006: 65 ms to steal Linux `dmcrypt` AES key used for hard-disk encryption

NaCl Security: No secret load addresses

- Osvik, Shamir, and Tromer in 2006: 65 ms to steal Linux `dmcrypt` AES key used for hard-disk encryption
- Attack background:
  - Most AES implementations use lookup tables
  - Secret AES key influences load addresses
  - Load addresses influence cache state
  - Cache state influences measurable timings
  - Use timing measurements to compute the key

NaCl Security: No secret load addresses

- Osvik, Shamir, and Tromer in 2006: 65 ms to steal Linux `dmcrypt` AES key used for hard-disk encryption
- Attack background:
  - Most AES implementations use lookup tables
  - Secret AES key influences load addresses
  - Load addresses influence cache state
  - Cache state influences measurable timings
  - Use timing measurements to compute the key
- Most cryptographic libraries still use lookup tables but add "countermeasures"
- Obscuring the influence on timings is not very confidence inspiring

NaCl Security: No secret load addresses

- Osvik, Shamir, and Tromer in 2006: 65 ms to steal Linux `dmcrypt` AES key used for hard-disk encryption
- Attack background:
  - Most AES implementations use lookup tables
  - Secret AES key influences load addresses
  - Load addresses influence cache state
  - Cache state influences measurable timings
  - Use timing measurements to compute the key
- Most cryptographic libraries still use lookup tables but add "countermeasures"
- Obscuring the influence on timings is not very confidence inspiring
- **NaCl systematically avoids all loads from addresses that depend on secret data**

NaCl Security: No secret branch conditions

- Brumley and Tuveri in 2011: A few minutes to steal OpenSSL ECDSA key

NaCl Security: No secret branch conditions

- Brumley and Tuveri in 2011: A few minutes to steal OpenSSL ECDSA key
- Attack background:
  - Branch conditions in scalar multiplication depend on key bits
  - Branch conditions influence timings
  - Use timing measurements to compute the key

NaCl Security: No secret branch conditions

- Brumley and Tuveri in 2011: A few minutes to steal OpenSSL ECDSA key
- Attack background:
  - Branch conditions in scalar multiplication depend on key bits
  - Branch conditions influence timings
  - Use timing measurements to compute the key
- Most cryptographic software has such data flow from secret data to branch conditions
- Example: `memcmp` to verify IPsec MACs

NaCl Security: No secret branch conditions

- Brumley and Tuveri in 2011: A few minutes to steal OpenSSL ECDSA key
- Attack background:
  - Branch conditions in scalar multiplication depend on key bits
  - Branch conditions influence timings
  - Use timing measurements to compute the key
- Most cryptographic software has such data flow from secret data to branch conditions
- Example: `memcmp` to verify IPsec MACs
- **NaCl systematically avoids all branch conditions that depend on secret data**

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:
  - X25519 for Diffie-Hellman key exchange

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:
  - X25519 for Diffie-Hellman key exchange
  - Ed25519 for Signatures

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:
  - X25519 for Diffie-Hellman key exchange
  - Ed25519 for Signatures
  - Salsa20 for stream encryption

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:
  - X25519 for Diffie-Hellman key exchange
  - Ed25519 for Signatures
  - Salsa20 for stream encryption
  - Poly1305 for one-time secret-key authentication

More NaCl Security

- No padding oracles: **NaCl does not decrypt unless ciphertext passes MAC verification**
- No unnecessary randomness: **NaCl uses deterministic signing**
- Centralizing randomness: **NaCl uses RNG from the OS**
- Conservative choice of primitives:
  - X25519 for Diffie-Hellman key exchange
  - Ed25519 for Signatures
  - Salsa20 for stream encryption
  - Poly1305 for one-time secret-key authentication
- At least 128 bits of security
- Easy to implement **efficiently and securely**

NaCl Speed

NaCl Speed

- **NaCl offers exceptionally high speeds, keeps up with the network**
- NaCl operations per second on AMD Phenom II X6 1100T for any reasonable packet size:
  - > 80000 `crypto_box`
  - > 80000 `crypto_box_open`
  - > 70000 `crypto_sign_open`
  - > 180000 `crypto_sign`
- Handles arbitrary packet floods up to ≈ 30 Mbps per CPU, depending on protocol
- Much faster than, e.g., TLS from OpenSSL

Even higher NaCl Speed

- Pure secret-key crypto for any packet size, 80000 packets of 1500 bytes fill up a 1 Gbps link

Even higher NaCl Speed

- Pure secret-key crypto for any packet size, 80000 packets of 1500 bytes fill up a 1 Gbps link
- Pure secret-key crypto for many packets from the same public key: split `crypto_box` into `crypto_box_beforenm` and `crypto_box_afternm`

Even higher NaCl Speed

- Pure secret-key crypto for any packet size, 80000 packets of 1500 bytes fill up a 1 Gbps link
- Pure secret-key crypto for many packets from the same public key: split `crypto_box` into `crypto_box_beforenm` and `crypto_box_afternm`
- Very fast rejection of forged packets under known public keys

Even higher NaCl Speed

- Pure secret-key crypto for any packet size, 80000 packets of 1500 bytes fill up a 1 Gbps link
- Pure secret-key crypto for many packets from the same public key: split `crypto_box` into `crypto_box_beforenm` and `crypto_box_afternm`
- Very fast rejection of forged packets under known public keys
- Fast batch signature verification: doubling verification speed

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**
- Optimize algorithms *across* primitives to reuse more code

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**
- Optimize algorithms *across* primitives to reuse more code
- Memory access is uncached: secret load addresses are not a problem!

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**
- 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

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**
- 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)

Introducing μNaCl

- Joint work with Michael Hutter: NaCl for embedded microcontrollers
- First target: **AVR ATmega**
- 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)
- Adresses have only 16 bits, so restrict message length to $2^{16} - 1$ (avoid expensive arithmetic on 64-bit integers)

AVR NaCl speeds and sizes

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 speeds and sizes

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

How about hardware?

- Joint work with Michael Hutter, Jürgen Schilling, and Wolfgang Wieser: Tiny ASIC for `crypto_box`
- Target application: WISP nodes
- Optimize for low power, not low energy

How about hardware?

- Joint work with Michael Hutter, Jürgen Schilling, and Wolfgang Wieser: Tiny ASIC for `crypto_box`
- Target application: WISP nodes
- Optimize for low power, not low energy
- Constant run-time implementation
- 32-bit ASIP

How about hardware?

- Joint work with Michael Hutter, Jürgen Schilling, and Wolfgang Wieser: Tiny ASIC for `crypto_box`
- Target application: WISP nodes
- Optimize for low power, not low energy
- Constant run-time implementation
- 32-bit ASIP
- Results:
  - 14.6–18.0kGE
  - 40–70 μW (half of power is spent for RAM)
  - 53.4–82.6ns (12–18MHz)
  - 811 170–3 455 394 cycles for DH

More around NaCl

- Bernstein, Schwabe, 2012:
  NaCl on ARM+NEON
  https://cryptojedi.org/crypto/#neoncrypto

More around NaCl

- Bernstein, Schwabe, 2012:
  NaCl on ARM+NEON
  https://cryptojedi.org/crypto/#neoncrypto
- Chen, Hsu, Lin, Schwabe, Tsai, Wang, Yang, Yang, 2014:
  Verify core of Curve25519 assembly implementation
  https://cryptojedi.org/crypto/#verify25519

More around NaCl

- Bernstein, Schwabe, 2012:
  NaCl on ARM+NEON
  https://cryptojedi.org/crypto/#neoncrypto
- Chen, Hsu, Lin, Schwabe, Tsai, Wang, Yang, Yang, 2014:
  Verify core of Curve25519 assembly implementation
  https://cryptojedi.org/crypto/#verify25519
- Hutter, Schwabe, 2015:
  Faster multiprecision multiplication on AVR
  https://cryptojedi.org/crypto/#avrmul

More around NaCl

- Bernstein, Schwabe, 2012:
  NaCl on ARM+NEON
  https://cryptojedi.org/crypto/#neoncrypto
- Chen, Hsu, Lin, Schwabe, Tsai, Wang, Yang, Yang, 2014:
  Verify core of Curve25519 assembly implementation
  https://cryptojedi.org/crypto/#verify25519
- Hutter, Schwabe, 2015:
  Faster multiprecision multiplication on AVR
  https://cryptojedi.org/crypto/#avrmul
- Düll, Haase, Hinterwälder, Hutter, Paar, Sánchez, Schwabe, 2015:
  Faster Curve25519 on AVR ATmega, TI MSP430, ARM Cortex-M0
  https://cryptojedi.org/crypto/#mu25519

More around NaCl

- Bernstein, Schwabe, 2012:
  NaCl on ARM+NEON
  https://cryptojedi.org/crypto/#neoncrypto
- Chen, Hsu, Lin, Schwabe, Tsai, Wang, Yang, Yang, 2014:
  Verify core of Curve25519 assembly implementation
  https://cryptojedi.org/crypto/#verify25519
- Hutter, Schwabe, 2015:
  Faster multiprecision multiplication on AVR
  https://cryptojedi.org/crypto/#avrmul
- Düll, Haase, Hinterwälder, Hutter, Paar, Sánchez, Schwabe, 2015:
  Faster Curve25519 on AVR ATmega, TI MSP430, ARM Cortex-M0
  https://cryptojedi.org/crypto/#mu25519
- Ongoing work together with Bernstein:
  Full verification of Curve25519 (and generally ECC)
  https://cryptojedi.org/crypto/#gfverif (online soon)

NaCl online

http://nacl.cr.yp.to

http://munacl.cryptojedi.org

http://cryptojedi.org/crypto/#naclhw

- No license: NaCl is in the public domain
- No patents that we are aware of