NaCl: Cryptography for the Internet

Peter Schwabe

Radboud University Nijmegen, The Netherlands

Joint work with Dan Bernstein and Tanja Lange

January 21, 2013

Workshop on Cryptography for the Internet, Tenerife, Spain

Why are we here?

- Various well understood algorithms, e.g. AES-128, RSA-2048, SHA-2, SHA-3 etc.
- Various implementations of these algorithms, bundled in libraries (e.g., OpenSSL)
- Applications simply use the libraries and the world (i.e., the Internet) is safe

Why are we here?

- Various well understood algorithms, e.g. AES-128, RSA-2048, SHA-2, SHA-3 etc.
- Various implementations of these algorithms, bundled in libraries (e.g., OpenSSL)
- Applications simply use the libraries and the world (i.e., the Internet) is safe

Two answers

- The above is wrong (I hope everybody here agrees)

Why are we here?

- Various well understood algorithms, e.g. AES-128, RSA-2048, SHA-2, SHA-3 etc.
- Various implementations of these algorithms, bundled in libraries (e.g., OpenSSL)
- Applications simply use the libraries and the world (i.e., the Internet) is safe

Two answers

- The above is wrong (I hope everybody here agrees)
- "Crypto for 2020" not only needs to fix existing problems but anticipate future ones

NaCl: A new cryptographic library

- Networking and Cryptography library (NaCl, pronounced “salt”)
- Aim: Fix the problems of crypto for the Internet
- 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: A new cryptographic library

- Networking and Cryptography library (NaCl, pronounced “salt”)
- Aim: Fix the problems of crypto for the Internet
- 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)

This talk

- Introduce NaCl

NaCl: A new cryptographic library

- Networking and Cryptography library (NaCl, pronounced “salt”)
- Aim: Fix the problems of crypto for the Internet
- 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)

This talk

- Introduce NaCl
- Topics I would like to discuss in the context of NaCl

Space shuttles vs. 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*

Protecting Internet communication . . .

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

. . . with the space-shuttle approach

- 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
- . . .

. . . with the space-shuttle approach

- 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.

. . . with the elevator approach

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

. . . with the elevator approach

```
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`

. . . with the elevator approach

```
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

. . . with the elevator approach

```
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

. . . with the elevator approach

```
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)
```

. . . with the elevator approach

```
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`

Back to space-shuttles and elevators: Security.

"About two percent of the manned launch/reentry attempts have killed their crew"
http://en.wikipedia.org/wiki/List_of_spaceflight-related_accidents_and_incidents

Back to space-shuttles and elevators: Security.

"About two percent of the manned launch/reentry attempts have killed their crew"
http://en.wikipedia.org/wiki/List_of_spaceflight-related_accidents_and_incidents

"the only known free-fall incident in a modern cable-borne elevator happened in 1945 when a B-25 bomber struck the Empire State Building in fog"
http://en.wikipedia.org/wiki/Elevator

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**

NaCl Security: No padding oracles

- Bleichenbacher in 1998: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.

NaCl Security: No padding oracles

- Bleichenbacher in 1998: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.
- Attack background:
  - SSL first inverts RSA, then checks for PKCS padding (which many forgeries have)
  - Subsequent processing applies more serious integrity checks
  - Server responses reveal pattern of PKCS forgeries
  - Pattern reveals plaintext

NaCl Security: No padding oracles

- Bleichenbacher in 1998: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.
- Attack background:
  - SSL first inverts RSA, then checks for PKCS padding (which many forgeries have)
  - Subsequent processing applies more serious integrity checks
  - Server responses reveal pattern of PKCS forgeries
  - Pattern reveals plaintext
- Typical protection: try to hide differences between padding checks and subsequent integrity checks
- Hard to get right; see, e.g., Crypto 2012 paper by Bardou, Focardi, Kawamoto, Steel, and Tsay

NaCl Security: No padding oracles

- Bleichenbacher in 1998: Decrypt SSL RSA ciphertext by observing server responses to $\approx 10^6$ variants of ciphertext.
- Attack background:
  - SSL first inverts RSA, then checks for PKCS padding (which many forgeries have)
  - Subsequent processing applies more serious integrity checks
  - Server responses reveal pattern of PKCS forgeries
  - Pattern reveals plaintext
- Typical protection: try to hide differences between padding checks and subsequent integrity checks
- Hard to get right; see, e.g., Crypto 2012 paper by Bardou, Focardi, Kawamoto, Steel, and Tsay
- **NaCl does not decrypt unless ciphertext passes MAC verification**
- **MAC verification in NaCl rejects forgeries in constant time**

NaCl Security: Centralizing randomness

- Bello in 2008: Debian/Ubuntu OpenSSL keys have only 15 bits of entropy
- Debian developer had removed on line of randomness-generating code

NaCl Security: Centralizing randomness

- Bello in 2008: Debian/Ubuntu OpenSSL keys have only 15 bits of entropy
- Debian developer had removed on line of randomness-generating code
- **NaCl uses `/dev/urandom`, the OS random-number generator**
- Reviewing this code is much more tractable than reviewing separate RNG in every library

NaCl Security: No unnecessary randomness

- "Bushing", Cantero, Boessenkool, Peter in 2010: Sony ignored ECDSA requirement of new randomness for each signature
- Signatures leaked PlayStation 3 code-signing key

NaCl Security: No unnecessary randomness

- "Bushing", Cantero, Boessenkool, Peter in 2010: Sony ignored ECDSA requirement of new randomness for each signature
- Signatures leaked PlayStation 3 code-signing key
- **NaCl uses *deterministic*** `crypto_box` **and** `crypto_sign`
- Also simplifies testing: NaCl uses automated test battery by eBACS (ECRYPT Benchmarking of Cryptographic Systems)

NaCl Security: Conservative choice of primitives

- Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, de Weger in 2008: rogue CA certificate, exploiting MD5 weakness
- "Flame" in 2012: New MD5 attack

NaCl Security: Conservative choice of primitives

- Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, de Weger in 2008: rogue CA certificate, exploiting MD5 weakness
- "Flame" in 2012: New MD5 attack
- By 1996 Dobbertin and Preneel were calling for MD5 to be scrapped

NaCl Security: Conservative choice of primitives

- Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, de Weger in 2008: rogue CA certificate, exploiting MD5 weakness
- "Flame" in 2012: New MD5 attack
- By 1996 Dobbertin and Preneel were calling for MD5 to be scrapped
- Many applications today use RSA-1024 (Google SSL, Tor, DNSSEC)
- Shamir and Tromer in 2003: RSA-1024 is breakable (1 year, $\approx 10^7$ USD)
- Reaction by NIST and RSA labs: Move to RSA-2048 by 2010

NaCl Security: Conservative choice of primitives

- Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, de Weger in 2008: rogue CA certificate, exploiting MD5 weakness
- "Flame" in 2012: New MD5 attack
- By 1996 Dobbertin and Preneel were calling for MD5 to be scrapped
- Many applications today use RSA-1024 (Google SSL, Tor, DNSSEC)
- Shamir and Tromer in 2003: RSA-1024 is breakable (1 year, $\approx 10^7$ USD)
- Reaction by NIST and RSA labs: Move to RSA-2048 by 2010
- **NaCl pays attention to cryptanalysis and makes very conservative choices**
- **Primitives in NaCl all offer 128 bits of security**

You might think that elevators are slow. . .

- Typical reason for low-security crypto or no crypto: speed
- For example, DNSSEC on using RSA-1024:

  "tradeoff between the risk of key compromise and performance. . . "

You might think that elevators are slow...

- Typical reason for low-security crypto or no crypto: speed
- For example, DNSSEC on using RSA-1024:

  > *"tradeoff between the risk of key compromise and performance..."*

- **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

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

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
- Also fast on mobile devices: See our CHES 2012 paper "NEON crypto"

NaCl online

http://nacl.cr.yp.to

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

Topics/Questions I'd like to discuss

- Is the "elevator approach" the right one to secure the Internet?
- What other functionalities (elevator buttons) are required?
- What important crypto-layer problems are not addressed by NaCl?

Topics/Questions I'd like to discuss

- Is the "elevator approach" the right one to secure the Internet?
- What other functionalities (elevator buttons) are required?
- What important crypto-layer problems are not addressed by NaCl?
- Deployment. . .

Topics/Questions I'd like to discuss

- Is the "elevator approach" the right one to secure the Internet?
- What other functionalities (elevator buttons) are required?
- What important crypto-layer problems are not addressed by NaCl?
- Deployment...
- NaCl for embedded devices
- Side-channel-protection requirements

Topics/Questions I'd like to discuss

- Is the "elevator approach" the right one to secure the Internet?
- What other functionalities (elevator buttons) are required?
- What important crypto-layer problems are not addressed by NaCl?
- Deployment...
- NaCl for embedded devices
- Side-channel-protection requirements
- Importance of correctness proofs

Topics/Questions I'd like to discuss

- Is the "elevator approach" the right one to secure the Internet?
- What other functionalities (elevator buttons) are required?
- What important crypto-layer problems are not addressed by NaCl?
- Deployment...
- NaCl for embedded devices
- Side-channel-protection requirements
- Importance of correctness proofs
- Importance of post-quantum NaCl