NaCl – Networking and Cryptography library

Peter Schwabe

Eindhoven University of Technology

2009-11-30

10th SPAN Workshop

Credits

- Work presented in this talk is mostly not my own work
- Responsible for NaCl are Daniel J. Bernstein and Tanja Lange
- Several other people contributing including me
- Thanks to NSF ITR-0716498
- Thanks to EU FP7 IST-216499 CACE

Introduction – Security and Cryptography

- Lots of applications in IT security rely on cryptography
- Two possibilities when they need crypto:
  - Develop your own primitives and protocols
  - Choose well-known and studied primitives

Introduction – Security and Cryptography

- Lots of applications in IT security rely on cryptography
- Two possibilities when they need crypto:
  - Develop your own primitives and protocols
  - Choose well-known and studied primitives
- Assuming the second, you have another choice:
  - Implement yourself
  - Take existing implementation (from a library)

Introduction – Security and Cryptography

- Lots of applications in IT security rely on cryptography
- Two possibilities when they need crypto:
  - Develop your own primitives and protocols
  - Choose well-known and studied primitives
- Assuming the second, you have another choice:
  - Implement yourself
  - Take existing implementation (from a library)
- Usual "best case": Use something like OpenSSL

Introduction – Security and Cryptography

- Lots of applications in IT security rely on cryptography
- Two possibilities when they need crypto:
  - Develop your own primitives and protocols
  - Choose well-known and studied primitives
- Assuming the second, you have another choice:
  - Implement yourself
  - Take existing implementation (from a library)
- Usual "best case": Use something like OpenSSL
- In the following: Use OpenSSL as example to show how NaCl improves upon other libraries
- Some examples are specific to OpenSSL, others are not

Things that go wrong – Part I: Speed

Technische Universiteit Eindhoven University of Technology

- Example: OpenSSL's AES implementation takes ≈ 18.3 cycles/byte on an Intel Core 2 Q6600
- ... and ≈ 14.3 on a Core 2 Q9550
- Speed records (for parallel modes): 9.32 and 7.59 cycles/byte respectively
- Almost a factor of 2 faster!

Things that go wrong – Part I: Speed

- Example: OpenSSL's AES implementation takes ≈ 18.3 cycles/byte on an Intel Core 2 Q6600
- … and ≈ 14.3 on a Core 2 Q9550
- Speed records (for parallel modes): 9.32 and 7.59 cycles/byte respectively
- Almost a factor of 2 faster!

Is speed that important?

- Example: Truecrypt moved from AES implementation in C to Assembly, reason: speed
- Another example: Have you ever tried to access `https://google.nl`?

Things that go wrong – Part I: Speed

- Example: OpenSSL's AES implementation takes ≈ 18.3 cycles/byte on an Intel Core 2 Q6600
- ... and ≈ 14.3 on a Core 2 Q9550
- Speed records (for parallel modes): 9.32 and 7.59 cycles/byte respectively
- Almost a factor of 2 faster!

Is speed that important?

- Example: Truecrypt moved from AES implementation in C to Assembly, reason: speed
- Another example: Have you ever tried to access `https://google.nl`?
- ... You will be redirected to `http://google.nl`
- Reason: Crypto is too expensive (too slow!)

Things that go wrong – Part I: Speed

Technische Universiteit Eindhoven University of Technology

- Example: OpenSSL's AES implementation takes ≈ 18.3 cycles/byte on an Intel Core 2 Q6600
- ... and ≈ 14.3 on a Core 2 Q9550
- Speed records (for parallel modes): 9.32 and 7.59 cycles/byte respectively
- Almost a factor of 2 faster!

Is speed that important?

- Example: Truecrypt moved from AES implementation in C to Assembly, reason: speed
- Another example: Have you ever tried to access `https://google.nl`?
- ... You will be redirected to `http://google.nl`
- Reason: Crypto is too expensive (too slow!)

$\Rightarrow$ A crypto library should offer the best possible speed for any given primitive and any given platform!

NaCl Part I: Speed

- How do we measure the speed of an implementation?
- Answer: SUPERCOP – **S**ystem for **U**nified **P**erformance **E**valuation **R**elated to **C**ryptographic **O**perations and **P**rimitives
- Benchmarking suite run by Daniel J. Bernstein and Tanja Lange
- On each platform: Compiles each implementation of each primitive with a huge variety of compiler options
- Checks compatibility with a a reference implementation
- Measures speed for different input lengths (if applicable)
- Currently contains benchmarking results from > 100 computers

NaCl Part I: Speed

- SUPERCOP and NaCl are using the same API
- They are also using the same build techniques
- On each computer:
  - Compile each implementation ...
  - of each primitive ...
  - with all possible (reasonable) compiler options ...
  - Pick the fastest one ...
  - Link all these fastest primitives together to the NaCl library
- Of course this still requires fast implementations
- Currently several speed-record-setting implementations are part of NaCl (or to be integrated)

Part II: Usability

Technische Universiteit **Eindhoven** University of Technology

- Let's try to encrypt and authenticate a given message with a given symmetric key and a given nonce
- Message: `char *m = "This is the message";`
- Key: `uint8_t key[32] = {0x00, 0x01, 0x02, ..., 0x1f};`
- Nonce: `uint8_t nonce[32] = {0x00, 0x00, ..., 0x00};`

Part II: Usability

- Let's try to encrypt and authenticate a given message with a given symmetric key and a given nonce
- Message: `char *m = "This is the message";`
- Key: `uint8_t key[32] = {0x00, 0x01, 0x02, ..., 0x1f};`
- Nonce: `uint8_t nonce[32] = {0x00, 0x00, ..., 0x00};`

...Code examples...

Part II: Usability

- Let's try to encrypt and authenticate a given message with a given symmetric key and a given nonce
- Message: `char *m = "This is the message";`
- Key: `uint8_t key[32] = {0x00, 0x01, 0x02, ..., 0x1f};`
- Nonce: `uint8_t nonce[32] = {0x00, 0x00, ..., 0x00};`

...Code examples...

- Verification + Decryption is pretty much the inverse
- For NaCl the function is called `crypto_secretbox_open`

Things that go wrong – Part III: Security
Choosing primitives

- From the example before: We chose AES256-CBC and HMAC-SHA256
- Why didn't we choose DES and HMAC-MD4?
- Why should *we* have to make the decision at all?
- The library is developed by crypto experts
- Why not let the experts choose what's best for "encrypt and authenticate"
- In particular if algorithms such as DES and MD4 are still in the library!

NaCl Part III: Security

Choosing primitives

Technische Universiteit **Eindhoven** University of Technology

- NaCl only contains high-security primitives
- No 80-bit security primitives
- High level functions such as `crypto_box`, `crypto_secretbox`, `crypto_scalarmult`, `crypto_hash`
- Underlying primitives chosen by experts
- It is still possible to give the primitives explicitely
- For example: Use `crypto_secretbox_aes256hmacsha512`

Idea of timing attacks

If execution time depends on secret data an attacker can deduce information by measuring the execution time

Examples for such timing variations

- Innput dependent branches (branch prediction)
- Loading from secret positions
  - Loads take different time depending on whether data is in cache
  - Attacker can overwrite certain cache lines
  - Check whether crypto implementation loaded from these lines
  - Remote attacks are also possible

All cryptographic libraries (I know) are vulnerable to such attacks!

NaCl Part III: Security

Preventing timing attacks

Technische Universiteit **Eindhoven** University of Technology

- In NaCl by default all implementations are constant time
- No secret-input-dependent branches, e.g. replace:

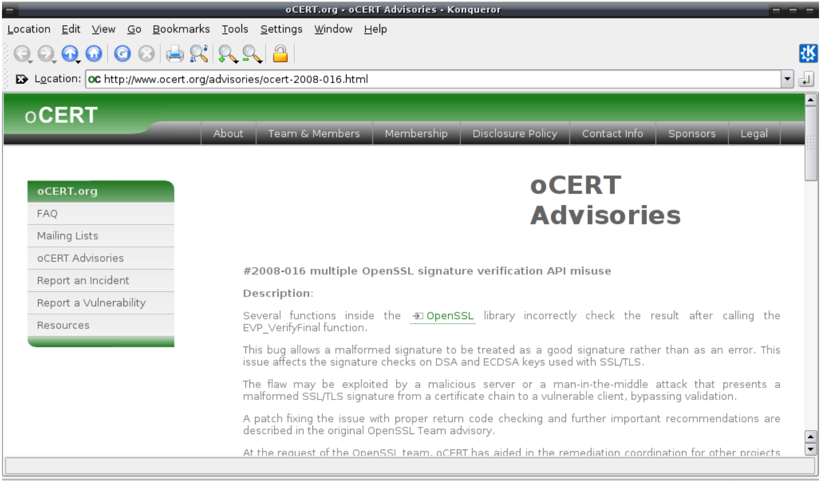
  ```
  if(a) b = c;
  else b = d;
  ```

  by

  ```
  b = a*c + (1-a)*d;
  ```

- No loads indexed by secret data by using techniques such as bitslicing
- Don't use `strcmp` to verify validity of auth tags
- If non-constant-time implementations are faster you can *choose* to use them
- Again, the default is: constant-time implementations!

Things that go wrong – Part IV: Bugs



Things that go wrong – Part IV: Bugs

"This bug allows a malformed signature to be treated as a good signature rather than as an error. This issue affects the signature checks on DSA and ECDSA keys used with SSL/TLS. The flaw may be exploited by a malicious server or a man-in-the-middle attack that presents a malformed SSL/TLS signature from a certificate chain to a vulnerable client, bypassing validation."

NaCl – Part IV: Avoiding Bugs

How can we avoid such bugs?

- Accept any byte array of appropriate length as valid input
- Systematic testing:
- In NaCl every implementation is checked during the build process
- Tests to ensure functionality, e.g:
  - Decryption is the inverse of encryption
  - Operations don't overwrite input
  - Extra bytes are cleared
  - ...
- Compatibility tests of different implementations of the same primitive
- Other groups within CACE are working on formal verification

Final remarks

Technische Universiteit **Eindhoven** University of Technology

- NaCl is in development, some primitives are not implemented yet
- No digital signatures yet
- No network functionality yet (although prototypes currently used in breaking ECC2K-130)
- All code is in public domain

Library: `http://nacl.cace-project.eu`
Benchmarking: `http://bench.cr.yp.to/supercop.html`
CACE Project: `http://cace-project.eu`