The security impact of a new cryptographic library

D. J. Bernstein, U. Illinois Chicago

Joint work with:

Tanja Lange, T. U. Eindhoven Peter Schwabe, National Taiwan U.



http://xkcd.com/538/

AES-128, RSA-2048, etc. are widely accepted standards.

Obviously infeasible to break by best attacks in literature.

Implementations are available in public cryptographic libraries such as OpenSSL.

Common security practice is to use those implementations.

AES-128, RSA-2048, etc. are widely accepted standards.

Obviously infeasible to break by best attacks in literature.

Implementations are available in public cryptographic libraries such as OpenSSL.

Common security practice is to use those implementations.

But cryptography is still a disaster! Complete failures of confidentiality and integrity. We have designed+implemented a new cryptographic library, NaCl ("salt"), to address the underlying problems.

nacl.cace-project.eu,
nacl.cr.yp.to: source
and extensive documentation.

Acknowledgments: code contributions from Matthew Dempsky (Mochi Media), Niels Duif (Eindhoven), Emilia Käsper (Leuven), Adam Langley (Google), Bo-Yin Yang (Academia Sinica).

Most of the Internet is cryptographically unprotected. Primary goal of NaCI: Fix this.

Main task: public-key authenticated encryption.

Alice has a message m for Bob.

Uses Bob's public key and Alice's secret key to compute authenticated ciphertext *c*. Sends *c* to Bob.

Bob uses Alice's public key and Bob's secret key to verify and recover *m*. Alice using a typical cryptographic library:

Generate random AES key. Use AES key to encrypt packet. Hash encrypted packet. Read RSA key from wire format. Use key to sign hash. Read Bob's key from wire format. Use key to encrypt signature etc. Convert to wire format.

Plus more code: 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)

32-byte secret key sk. 32-byte public key pk. 24-byte nonce n. c is 16 bytes longer than m. All objects are C++std::string variables represented in wire format, ready for storage/transmission.

C NaCI: similar, using pointers; no memory allocation, no failures. Bob verifying, decrypting: m=crypto_box_open(c,n,pk,sk)

Initial key generation:

pk = crypto_box_keypair(&sk)

Bob verifying, decrypting: m=crypto_box_open(c,n,pk,sk)

Initial key generation:
pk = crypto_box_keypair(&sk)

Can instead use **signatures** for public messages:

pk = crypto_sign_keypair(&sk)
64-byte secret key,
32-byte public key.

sm = crypto_sign(m,sk)
64 bytes overhead.

m = crypto_sign_open(sm,pk)

"This sounds too simple! Don't applications need more?" "This sounds too simple! Don't applications need more?"

Examples of applications using NaCl's crypto_box:

Several implementations of DNSCurve, high-security authenticated encryption for DNS queries; deployed by OpenDNS.

QuickTun, VPN from Ivo Smits.

Ethos, OS from Jon Solworth.

Prototype implementation of CurveCP: high-security cryptographic version of TCP.

No secret load addresses

2005 Osvik–Shamir–Tromer: 65ms to steal Linux AES key used for hard-disk encryption. Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms to compute influence⁻¹. Most cryptographic libraries still use secret load addresses but add "countermeasures" intended to obscure influence upon the CPU cache state. Not confidence-inspiring; likely to be breakable. Most cryptographic libraries still use secret load addresses but add "countermeasures" intended to obscure influence upon the CPU cache state. Not confidence-inspiring; likely to be breakable.

NaCl systematically avoids *all* loads from addresses that depend on secret data. Eliminates this type of disaster.

2010 Langley ctgrind: verify this automatically.

No secret branch conditions

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing:

e.g., memcmp for IPsec MACs.

No secret branch conditions

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing:

e.g., memcmp for IPsec MACs.

NaCl systematically avoids *all* branch conditions that depend on secret data. Eliminates this type of disaster.

No padding oracles

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

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 defense strategy: try to hide differences between padding checks and subsequent integrity checks.

Hard to get this right: e.g. 2009 Albrecht–Paterson–Watson recovered some SSH plaintext. Typical defense strategy: try to hide differences between padding checks and subsequent integrity checks.

Hard to get this right: e.g. 2009 Albrecht–Paterson–Watson recovered some SSH plaintext.

NaCl does not decrypt unless message is authenticated. Verification procedure rejects all forgeries in constant time. Attacks are further constrained by per-nonce key separation and standard nonce handling.

Centralizing randomness

2008 Bello: Debian/Ubuntu OpenSSL keys for 1.5 years had only 15 bits of entropy.

Debian developer had removed a subtle line of OpenSSL randomness-generating code.

Centralizing randomness

2008 Bello: Debian/Ubuntu OpenSSL keys for 1.5 years had only 15 bits of entropy.

Debian developer had removed a subtle line of OpenSSL randomness-generating code.

NaCl uses /dev/urandom, the OS random-number generator. Reviewing this kernel code is much more tractable than reviewing separate RNG code in every security library.

Avoiding unnecessary randomness

2010 Bushing–Marcan–Segher– Sven: Sony ignored ECDSA requirement of new randomness for each signature. ⇒ Signatures leaked PS3 code-signing key.

Avoiding unnecessary randomness

2010 Bushing–Marcan–Segher– Sven: Sony ignored ECDSA requirement of new randomness for each signature. ⇒ Signatures leaked PS3 code-signing key.

NaCl has *deterministic* crypto_box and crypto_sign. Randomness only for keypair. Eliminates this type of disaster.

Also simplifies testing. NaCl uses automated test battery from eBACS (ECRYPT Benchmarking of Cryptographic Systems).

Avoiding pure crypto failures

2008 Stevens–Sotirov– Appelbaum–Lenstra–Molnar– Osvik–de Weger exploited MD5 weaknesses to create rogue CA certificate.

Avoiding pure crypto failures

2008 Stevens–Sotirov– Appelbaum–Lenstra–Molnar– Osvik–de Weger exploited MD5 weaknesses to create rogue CA certificate.

Fact: By 1996, a few years after the introduction of MD5, Preneel and Dobbertin were calling for MD5 to be scrapped.

NaCl *pays attention to cryptanalysis* and makes very conservative choices of cryptographic primitives.

<u>Speed</u>

Crypto performance problems often lead users to reduce cryptographic security levels or give up on cryptography.

Example 1: Google SSL uses RSA-1024.

Security note: Analyses in 2003 concluded that RSA-1024 was breakable; e.g., 2003 Shamir–Tromer estimated 1 year, $\approx 10^7$ USD. RSA Labs and NIST response: Move to RSA-2048 by 2010. Example 2: Tor uses RSA-1024. Example 3: DNSSEC uses RSA-1024: "tradeoff between the risk of key compromise and performance..."

Example 4: OpenSSL continues to use secret AES load addresses.

Example 5:

https://sourceforge.net/account
is protected by SSL but

https://sourceforge.net/develop
redirects browser to

http://sourceforge.net/develop,
turning off the cryptography.

NaCl has no low-security options.

e.g. crypto_box always encrypts *and* authenticates. e.g. no RSA-1024;

not even RSA-2048.

NaCl has no low-security options.

e.g. crypto_box always encrypts *and* authenticates. e.g. no RSA-1024;

not even RSA-2048.

Remaining risk: Users find NaCl too slow \Rightarrow switch to low-security libraries or disable crypto entirely. NaCl has no low-security options.

e.g. crypto_box always
encrypts and authenticates.
e.g. no RSA-1024;
not even RSA-2048.

Remaining risk: Users find NaCl too slow \Rightarrow switch to low-security libraries or disable crypto entirely.

How NaCl avoids this risk: NaCl is exceptionally fast. Much faster than other libraries. *Keeps up with the network.*

NaCl operations per second for any common packet size, using AMD Phenom II X6 1100T CPU, 164€ on kieskeurig.nl: crypto_box: >80000. crypto_box_open: >80000. crypto_sign_open: >70000. crypto_sign: >180000. Putting this into perspective: 80000 50-byte packets/second would consume 32 Mbps plus per-packet overhead.

But wait, it's even faster!

Pure secret-key crypto
 for any packet size:
 80000 1500-byte packets/second
 fill up a 1 Gbps link.

2. Pure secret-key crypto for many packets from same public key, if application splits crypto_box into crypto_box_beforenm and crypto_box_afternm. Very fast rejection
 of forged packets
 under known public keys:
 no time spent on decryption.

(This doesn't help much for forgeries under *new* keys, but flooded server can continue providing fast service to *known* keys.)

 Fast batch verification, doubling speed of crypto_sign_open for valid signatures.

Cryptographic details

The main work we did: achieve these speeds *without* compromising security.

ECC, not RSA:

much stronger security record. Curve25519, not NSA/NIST curves: twist-security et al. Salsa20, not AES: much larger security margin. Poly1305, not HMAC: information-theoretic security. EdDSA, not ECDSA: collision-resilience et al.