Deploying high-security cryptography

Daniel J. Bernstein
University of Illinois at Chicago

http://cr.yp.to/talks.html
#2012.03.08-2

http://xkcd.com/538/
\texttt{nacl.crypt.to}: NaCl ("salt"), a new cryptographic library.

Core development team: Tanja Lange (Eindhoven); Peter Schwabe (Academia Sinica); me.

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

You’re a programmer.
Your program will send data through a dangerous link:

- WiFi at hotel, airport, coffee shop, etc.
- Firewalled corporate network with thousands of computers.
- Your own hard drive, communicating to you.

You’re worried about espionage, corruption, sabotage.

How to protect the data?
Often you’re happy with PGP, SSH, SSL, etc. Great!
Often you’re happy with PGP, SSH, SSL, etc.
Great!

But what do you do when the existing solutions aren’t good enough?
Need to program something.
Often you’re happy with PGP, SSH, SSL, etc.
Great!

But what do you do when the existing solutions aren’t good enough?
Need to program something.

Focus of this talk: using NaCl to deploy high-security cryptography in applications that don’t have cryptography yet.
Case study: Retrieving http://www.kuleuven.be.

1. Browser → web server:
   GET / HTTP/1.1
   Host: www.kuleuven.be

2. Web server → browser:
   HTTP/1.1 200 OK
   Date: Wed, 07 Mar 2012 10:23:23 GMT
   Server: Apache/1.3.37 (Unix) AuthMySQL/2.20 mod_ssl/2.8.28 OpenSSL/0.9.7e
   Content-Type: text/html etc.
These network packets have no security: no confidentiality, no integrity, no availability.

Network eavesdropper “Eve” easily sees and controls all packets on network.

Espionage: Eve sees data. Corruption: Eve modifies data. Sabotage: Eve kills connection (e.g., forging TCP RST packet).

How do we add security?
Actually, retrieving http://www.kuleuven.be has even more steps.

1. Browser → DNS server: Where is www.kuleuven.be?

2. DNS server → browser: www.kuleuven.be has IP address 134.58.64.12.

3. Browser → 134.58.64.12: GET / HTTP/1.1 etc.

4. 134.58.64.12 → browser: HTTP/1.1 200 OK etc.
Actually even more steps: proxies, redirects, images, etc.

1. Browser → DNS cache: 
Where is www.kuleuven.be?

2. DNS cache → DNS server: 
Where is www.kuleuven.be?

3. DNS server → DNS cache: 
www.kuleuven.be has IP address 134.58.64.12.

4. DNS cache → browser: 
www.kuleuven.be has IP address 134.58.64.12.
5. Browser → 134.58.64.12:
GET / HTTP/1.1 etc.

6. 134.58.64.12 → browser:
HTTP/1.1 302 Found
Location:
http://www.kuleuven.be
/kuleuven/ etc.

7. Browser → 134.58.64.12:
GET /kuleuven/ HTTP/1.1 etc.

8. 134.58.64.12 → browser:
HTTP/1.1 200 OK etc.

+ more servers, more packets . . .
Browser and kuleuven.be both support SSL.

Encrypts client’s HTTP requests; encrypts and authenticates server’s HTTP responses.

My top concern with SSL:
• Wide deployment is painful.

Other concerns:
• Doesn’t even try to stop DoS.
• Limited confidentiality.
• Too many trusted parties.
• Too complex to be secure.
“DNSCrypt”: New OpenDNS project using NaCl for easy-to-deploy confidentiality, integrity, availability of packets between browser ↔ DNS cache.


2012.02: “10’s of 1000’s more have downloaded DNSCrypt for Mac and are running it today.”
If Eve forges packet DNS cache → browser, DNSCrypt simply drops it, waits for correct packet.

Eve can still deny service by flooding the network—but floods cost much more, so Eve has fewer victims.

Clear availability benefit.
Eve still denies service by forging packets DNS cache ↔ DNS server or browser ↔ web server.

Fix: Protect those packets too! “DNSCurve” for DNS server, “HTTPCurve” for HTTP server. Have prototypes using NaCl.

Should end up reasonably easy to deploy confidentiality, integrity, availability for every Internet packet.
Why bother encrypting DNS if Eve sees 134.58.64.12?
Why bother encrypting DNS if Eve sees 134.58.64.12?

Answer 1: Some DNS names are secrets, effectively used for access control.
Why bother encrypting DNS if Eve sees 134.58.64.12?

Answer 1: Some DNS names are secrets, effectively used for access control.

Answer 2: cancer.webmd.com and fitness.webmd.com are on the same web server.
Why bother encrypting DNS if Eve sees 134.58.64.12?

Answer 1: Some DNS names are secrets, effectively used for access control.

Answer 2: cancer.webmd.com and fitness.webmd.com are on the same web server.

Answer 3: Use Tor.
Why bother encrypting DNS if Eve sees 134.58.64.12?

Answer 1: Some DNS names are secrets, effectively used for access control.

Answer 2: cancer.webmd.com and fitness.webmd.com are on the same web server.

Answer 3: Use Tor.

Answer 4: Encryption is practically free along with authentication. Making it optional would be pointless complexity.
Cryptographic library APIs

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.
Gutmann cryptlib library: “high-level interface” that “provides anyone with the ability to add strong security capabilities to an application in as little as half an hour, without needing to know any of the low-level details that make the encryption or authentication work.”

Alice using cryptlib: Look at the first code segment in the cryptlib manual (“the best way to illustrate what cryptlib can do”).
cryptCreateEnvelope(
    &cryptEnvelope,cryptUser,
    CRYPT_FORMAT_SMIME);
cryptSetAttributeString(
    cryptEnvelope,
    CRYPT_ENVINFO_RECIPIENT,
    recipientName,recipientNameLength
)
cryptPushData(cryptEnvelope,
    message,messageSize,
    &bytesIn);
cryptFlushData(cryptEnvelope);
cryptPopData(cryptEnvelope,
    encryptedMessage,encryptedSize,
    &bytesOut);
cryptDestroyEnvelope(cryptEnvelope)
Start with cryptInit.

Also check that each function returns CRYPT_OK.
(Page 35 of manual: the wrong code without checks is included “for clarity”.)

Also loop around cryptPushData, checking bytesIn. (Page 53 of manual.)

Data is encrypted without authentication.
Do more work to add signatures.
Alice using NaCl:

c = crypto_box(m, n, pk, sk)
Alice using NaCl:
\[ c = \text{crypto_box}(m,n,pk,sk) \]
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++ \texttt{std::string} variables represented in wire format, ready for storage/transmission.

If \texttt{crypto_box} runs out of memory, it raises an exception.
Bob verifying, decrypting:
\[ m = \text{crypto\_box\_open}(c, n, pk, sk) \]

Initial key generation:
\[ pk = \text{crypto\_box\_keypair}(\&sk) \]
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)
More languages

C: released.
Disadvantages: strings are passed as pointers and lengths; must check return value from crypto_*_open.
Advantages: No heap usage. Controlled stack usage. Most functions cannot fail.

C++: released.

Python: bindings online from Sean Lynch (Facebook), Adam Langley, Jan Mojzis; have integrated; testing.
To the extent possible, identical—purely functional—interface in every language.

Python:
```python
sm = crypto_sign(m, sk)
```

C++:
```cpp
sm = crypto_sign(m, sk)
```

C:
```c
crypto_sign(sm, &smlen, m, mlen, sk)
```
Security features

NaCl systematically avoids *all* array indices that depend on secret data.

NaCl systematically avoids *all* branch conditions that depend on secret data.

NaCl does not decrypt unless message is authenticated. Verification procedure rejects all forgeries in constant time.
NaCl has *deterministic* `crypto_box` and `crypto_sign`. Randomness only for keypair. Avoids PS3 signing disaster and many potential problems.

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

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

NaCl operations per second for any common packet size, using AMD Phenom II X6 1100T CPU, 189€ on kieskeurig.be:

- crypto_box: >80000.
- crypto_box_open: >80000.
- crypto_sign_open: >70000.
- crypto_sign: >180000.

Putting this into perspective: 80000 50-byte packets/second network usage is 32 Mbps plus per-packet overhead.
But wait, it’s even faster!

1. Pure secret-key crypto for longer packets: 
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
\[ c = \text{crypto\_box}(m,n,pk,sk) \]
into
\[ k = \text{crypto\_box\_beforenm}(pk,sk) \]
and then
\[ c = \text{crypto\_box\_afternm}(m,n,k). \]
3. 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.)

4. Fast batch verification, doubling speed of crypto_sign_open for valid signatures. (Not integrated yet.)
News: We’ve just finished initial NaCl smartphone optimizations.

800 MHz Cortex A8, 1 core, operations per second:

- crypto_box: 1500.
- crypto_box_open: 1500.
- crypto_sign_open: 1200.
- crypto_sign: 2100.

Secret-key crypto:
2.78 Gbps for authenticator.
1.14 Gbps for cipher.
Cryptographic details

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


Salsa20, not AES: much larger security margin.

Poly1305, not HMAC: information-theoretic security.

EdDSA, not ECDSA: collision-resilience et al.
Tasks outside NaCl

NaCl key generation relies completely on OS as a source of random bytes: e.g. /dev/urandom.

OS has much better access to entropy sources than any individual library does.

Reviewing centralized OS code is much easier than reviewing all the randomness-generation code in libraries that decide to do the job themselves.
NaCl relies on programmer to supply a nonce $n$: a message number, never reused.

One safe choice of nonce: 24 random bytes from the OS.

Or: 1 for first message, 2 for second message, etc. Smaller; faster; allows easy replay detection; but sometimes a privacy leak.
Protocol integration

Have an unprotected protocol. Want to apply NaCl. What to do?

How do DNSCrypt etc. use NaCl?

Easy: Before sending a packet, feed it to crypto_box. Transmit pk, n, c.

When a packet arrives, feed it to crypto_box_open, and throw it away if it’s bad.
How does client know that server will tolerate cryptographic packets?

DNSCrypt answer: OpenDNS will tolerate cryptographic packets.

DNSCurve answer: Look for DNSCurve indicator in name of the DNS server.

HTTPCurve answer: Look for DNS alias (CNAME) to an HTTPCurve indicator.
How does client find server’s public key?

DNSCrypt answer: Use DNS TXT lookup to retrieve signed short-term public key from server. Check signature under long-term OpenDNS public key included with DNSCrypt software.

DNSCurve answer: Look for DNSCurve public key in name of the DNS server.

HTTPCurve answer: Similar.