libsecp256k1

Optimized C library for ECDSA signatures and secret/public key operations on curve secp256k1.

This library is intended to be the highest quality publicly available library for cryptography on the secp256k1 curve. However, the primary focus of its development has been for usage in the Bitcoin system and usage unlike Bitcoin's may be less well tested, verified, or suffer from a less well thought out interface. Correct usage requires some care and consideration that the library is fit for your application's purpose.

Features:

- secp256k1 ECDSA signing/verification and key generation.
- Additive and multiplicative tweaking of secret/public keys.
- Serialization/parsing of secret keys, public keys, signatures.
- Constant time, constant memory access signing and public key generation.
- Derandomized ECDSA (via RFC6979 or with a caller provided function.)
- Very efficient implementation.
- Suitable for embedded systems.
- Optional module for public key recovery.
- Optional module for ECDH key exchange.
- Optional module for Schnorr signatures according to BIP-340 (experimental).

Experimental features have not received enough scrutiny to satisfy the standard of quality of this library but are made available for testing and review by the community. The APIs of these features should not be considered stable.

Implementation details

- General
  - No runtime heap allocation.
  - Extensive testing infrastructure.
  - Structured to facilitate review and analysis.
  - Intended to be portable to any system with a C89 compiler and uint64_t support.
  - No use of floating types.
  - Expose only higher level interfaces to minimize the API surface and improve application security. ("Be difficult to use insecurely.")
- Field operations
  - Optimized implementation of arithmetic modulo the curve's field size (2^256 - 0x1000003D1).
    - Using 5 52-bit limbs (including hand-optimized assembly for x86_64, by Diederik Huys).
    - Using 10 26-bit limbs (including hand-optimized assembly for 32-bit ARM, by Wladimir J. van der Laan).
- Scalar operations
  - Optimized implementation without data-dependent branches of arithmetic modulo the curve's order.
    - Using 4 64-bit limbs (relying on __int128 support in the compiler).
    - Using 8 32-bit limbs.
- Modular inverses (both field elements and scalars) based on safegcd with some modifications, and a variable-time variant (by Peter Dettman).

- Group operations
  - Point addition formula specifically simplified for the curve equation (y^2 = x^3 + 7).
  - Use addition between points in Jacobian and affine coordinates where possible.
  - Use a unified addition/doubling formula where necessary to avoid data-dependent branches.
  - Point/x comparison without a field inversion by comparison in the Jacobian coordinate space.
- Point multiplication for verification (a*P* + *b*G).
  - Use wNAF notation for point multiplicands.
  - Use a much larger window for multiples of G, using precomputed multiples.
  - Use Shamir's trick to do the multiplication with the public key and the generator simultaneously.
  - Use secp256k1's efficiently-computable endomorphism to split the P multiplicand into 2 half-sized ones.
- Point multiplication for signing
  - Use a precomputed table of multiples of powers of 16 multiplied with the generator, so general multiplication becomes a series of additions.
  - Intended to be completely free of timing sidechannels for secret-key operations (on reasonable hardware/toolchains)
    - Access the table with branch-free conditional moves so memory access is uniform.
    - No data-dependent branches
  - Optional runtime blinding which attempts to frustrate differential power analysis.
  - The precomputed tables add and eventually subtract points for which no known scalar (secret key) is known, preventing even an attacker with control over the secret key used to control the data internally.

Build steps

libsecp256k1 is built using autotools:

```
$ ./autogen.sh
$ ./configure
$ make
$ make check  # run the test suite
$ sudo make install  # optional
```

Test coverage

This library aims to have full coverage of the reachable lines and branches.

To create a test coverage report, configure with `--enable-coverage` (use of GCC is necessary):

```
$ ./configure --enable-coverage
```

Run the tests:

```
$ make check
```

To create a report, `gcovr` is recommended, as it includes branch coverage reporting:

```
$ gcovr --exclude 'src/bench*' --print-summary
```

To create a HTML report with coloured and annotated source code:

```
$ mkdir -p coverage
$ gcovr --exclude 'src/bench*' --html --html-details -o coverage/coverage.html
```

Benchmark

If configured with `--enable-benchmark` (which is the default), binaries for benchmarking the libsecp256k1 functions will be present in the root directory after the build.

To print the benchmark result to the command line:

```
$ ./bench_name
```

To create a CSV file for the benchmark result :

```
$ ./bench_name | sed '2d;s/ \{1,\}//g' > bench_name.csv
```

Reporting a vulnerability

See SECURITY.md