μNaClの32-bit ARM Cortex-M3への実装

西永　俊文†　　満保　雅浩‡

†金沢大学
920-1192 金沢市角間町

‡金沢大学
920-1192 金沢市角間町

あらまし IoT（Internert of Things）技術の普及に伴い，インターネットを介した脅威からモノを守るためのセキュリティ対策が急務となっている．モノの中にはマイコンを使った製品もあり，これらの製品は性能の問題より十分なセキュリティを確保するのが難しいため，マイコンで動作する軽量で高速かつ安全な暗号ライブラリが求められている．本論文では 8-bit AVR マイコン向け暗号ライブラリ AVR NaCl を 32-bit ARM Cortex-M3 マイコンに実装し，安全かつ高速にするための実装上の工夫を加える．そして，その実装を評価する．

Implementation of μNaCl on 32-bit ARM Cortex-M3

Toshifumi NISHINAGA†　　Masahiro MAMBO‡

†The University of Kanazawa.
Kakuma, Kanazawa, Ishikawa 852-8104, JAPAN
hogehoge@stu.kanazawa-u.ac.jp

‡The University of Kanazawa.
Kakuma, Kanazawa, Ishikawa 852-8104, JAPAN

Abstract By the deployment of Internet of Things, it has become unavoidable to take some security measure in embedded systems using microcontroller against threats through the network. Unfortunately, microcontrollers are not so powerful enough to execute standard security programs and need light-weight, high-speed and secure cryptographic libraries. In this paper, we implement an NaCl based cryptographic library on an ARM Cortex-M3 Microcontroller, where we put much effort in fast and secure implementation, and evaluate the implementation.

1　はじめに

様々なモノがインターネットに接続する IoT（Internet of Things）技術が，スマートフォンの普及，無線モジュールの高性能・低価格化に伴い，広がりつつある．

インターネットへ接続されるモノの中にはマイコンを用いた製品も含まれる．これらの製品をインターネットを介した攻撃から守るため十分なセキュリティの確保が望まれるが，マイコンは性能やコードサイズの問題から十分なセキュリティを確保するのが難しい．よって，十分なセキュリティを確保するため，マイコンで動作する軽量で高速かつ安全な暗号ライブラリが求められている．

現在主要なマイコンは，データ長が 8bit，16bit，32bit の 3 種であり，目的に応じて使い分けられている．中でも一番性能が高いのは 32bit マイコンである．32bit マイコンは一度に 32bit のデータを扱えるため，通信やセキュリティの処理を現段階では最も高速に行える．よって，インターネットに接続するマイコンとして，現段階では 32bit マイコンが好ましい．

32bit マイコンでは，ARM 社の開発した Cortex-M ファミリ マイクロコントローラー（以降，ARM マイコン）を載せたデバイスが，2013 年には 20 億台以上出荷 [1] され，広がりをみせている．加えて，高速プロトタイピングを支援する mbed プラットフォームの登場により，ARM マイコンを用いた高速プロトタイピングが簡単に行えるようになり，今後 ARM マイコンは全世界でさらに普及していくと推測される．

一方，軽量で高速かつ安全な暗号ライブラリを目指して設計された暗号ライブラリとして，Networking and Cryptography library(NaCl)[2] がある．このライブラリは高い安全性を実現するために非常に厳しい暗号方式の選択や実装が行われている．加えて，軽量で高速なコードとなるよう，様々な最適化が施されている．この NaCl は後にマイコン向けに最適化した μNaCl が開発され，μNaCl を 8bit AVR マイコン上に実装した AVR NaCl[3] が発表されている．しかし，AVR マイコンでネットワークを介する通信を行うには性能が足りず，更に AVR NaCl を用いた通信の暗号化を行うことは実用的ではないと考えられる．これに対して，32bit ARM マイコンは AVR マイコンに比べて非常に性能が高いため，この ARM マイコンをターゲットに μNaCl を実装することで，μNaCl の実用化が期待できる．また，8bit AVR マイコン用のコードを 32bit ARM マイコン用に最適化することで，コードサイズと実行サイクル数の大幅な削減を期待できる．

そこで本論文では，32bit マイコンにおいて軽量で高速かつ安全な暗号ライブラリを実装することを目標に，μNaCl を ARM Cortex-M3 マイコンへ実装する．その実装に際して，いくつかの高速化と安全性向上を図る．

本論文の構成は以下のようになっている．第 2 節では，本論文を理解する上で必要となる，暗号ライブラリ NaCl と μNaCl 及び ARM Cortex-M3 マイコンについての知識を述べる．第 3 節では，性能向上と安全のため本実装において NaCl に追加した手法の説明と，実装後の性能評価を行う．最後に第 4 節にて，本論文のまとめと今後の課題について述べる．

表 1: NaCl のプリミティブ

楕円曲線暗号	Curve25519[5]
ストリーム暗号	Salsa20[6]
Ec デジタル署名	Ed25519
メッセージ認証子	Poly1305[7]
ハッシュ関数	SHA512[8]

2 予備知識

2.1 Networking and Cryptography library(NaCl) と μNaCl

Networking and Cryptography library(NaCl)[4][2] は，OpenSSL の置き換えを目標として，使い勝手の良さ，高い安全性，高速性を満たすように Daniel J. Bernstein らが 2010 年にリリースしたライセンスフリーの暗号ライブラリである．

NaCl は表 1 のプリミティブを備え，プリミティブを容易に扱えるようにするために，プリミティブを組み合わせた API を提供している．

NaCl は暗号ライブラリに関する既知の脆弱性問題を生じにくい，以下 (1) から (3) の設計方針 [2, Section3] を採用している．

(1) ロードアドレスの秘匿

ロードアドレスの秘匿は，CPU キャッシュのヒット時とミス時のアクセス時間の差異に基づくタイミング攻撃を防ぐ対策である．この攻撃に対し，8bit AVR マイコンと ARM Cortex-M3 マイコンはどちらもキャッシュメモリを備えていないため，安全である．

(2) 秘密情報に依存する条件分岐の秘匿

分岐状態の秘匿は，秘密情報に依存した条件分岐に対して実行時間の差異を用いて攻撃する，タイミング攻撃への対策である．条件分岐に関するタイミング攻撃の原因として，鍵の値域に

依存した処理の省略 [9] や，分岐予測の成否 [10] がある．これらの攻撃に対し，NaCl では

- 秘密情報に依存する条件分岐を除去する
- 全てのループ回数を決定的にする

という対策を行い，AVR NaCl にも反映されている．加えて，8bit AVR マイコンの場合は，分岐予測ユニット及びパイプラインを備えていないため安全である．一方，ARM Cortex-M3 マイコンは3段のパイプラインと分岐予測ユニットを持つため，対策が必要である．

(3) 乱数の集中化と不必要な乱数利用の除去

2008 年，OpenSSL が予測可能な乱数を生成する問題（CVE-2008-0166）[11] が発覚した．この問題の原因は，OpenSSL の乱数生成コードへ行われた，誤った修正にある．NaCl はこのような問題を防ぐため，乱数生成機能を OS の乱数生成機能（例: /dev/urandom ）に一元化している [2, Section3 Centralizing randomness]．これにより，ライブラリごとに安全性を考慮する必要がなくなり，NaCl の乱数の安全性は OS の乱数生成機能の安全性により保証される．加えて，不必要な乱数の利用を控えることで，NaCl は乱数の安全性に起因する問題を減らしている [2, Section3 Avoiding unnecessary randomness]．

2.2 ARM Cortex-M3 マイクロコントローラ

ARM Cortex-M3 は 13 の 32bit 汎用レジスタ（r0-r12）と 3 つの特殊レジスタ（r13-r15）を持つ．r13-r15 は特殊レジスタであり，r13 はスタックポインタ（sp），r14 はリンクレジスタ (lr)，r15 はプログラムカウンタ（pc）となっている．

命令セットは 16bit または 32bit 命令長の Thumb2 を採用しており，この Thumb2 命令セットを用いることで，性能とエネルギー効率，コード密度をいずれも高くすることができる．

Cortex-M3 はハードウェア除算とシングルサイクル 32bit 乗算，飽和演算をサポートする．これらの命令とサイクルについて表 2 に示す．

表 2: 拡張命令

操作	説明	命令	サイクル数
乗算	32×32=32bit 乗算	mul	1
	符号（付き/なし）32×32=64bit 乗算	(s/u)mull	3 から 5
除算	符号（付き/なし）32bit 除算	(s/u)div	2 から 12
飽和	符号（付き/なし）	(s/u)sat	1

mull 命令ではソース値のサイズに応じて，div 命令では入力オペランドの先行の 1 と 0 の数に応じて，早期終了が使用される．そのため，mul 命令と div 命令のサイクル数は，共に可変となる．

更に，Cortex-M3 は 3 段のパイプラインと分岐予測ユニットを有し，これを効率よく用いることができれば，処理を高速化できる．

3 実装方式

AVR NaCl は最適化と安全性向上のため，暗号ライブラリの主要部は約 6000 行の AVR アセンブリ言語で書かれている．本論文ではこの AVR アセンブリコードを Thumb2 アセンブリ言語で書き直し，μNaCl が ARM Cortex-M3 上で動作するように実装する．加えて，Cortex-M3 の特徴に合わせて AVR NaCl の実装を修正し，高速化と安全性向上を実現する．

本実装手法の要点は，以下の 3 つである．

- 64bit 右シフト・ローテーション処理の最適化による SHA2-512 の高速化
- タイミング攻撃防止を目的とする実行時間可変乗算命令（umull）の除去
- SRAM の初期値をエントロピー源とする擬似乱数生成手法の採用

本実装は Cortex-M3 コアの LPC1768 を搭載した，mbed LPC1768 ボードを対象に行う．コンパイラは gcc(ARM/embedded-4_8-branch revision 208322) を用いる．

分岐状態の秘匿についての方針

8bit AVR マイコンは分岐予測ユニット及びパイプラインを持たないため，分岐予測の成否に起因する実行時間の差異を用いたタイミング攻撃は起こりえない．一方，ARM Cortex-M3 マイコンは 3 段のパイプラインと分岐予測ユニットを持つため，対策が必要である．この点について本実装では，2.1 節で述べた NaCl と同様の対策を行う．加えて，秘密情報に依存しない条件分岐についても，より高い安全性の確保と高速化のため，可能な限り条件分岐の除去を行う，

3.1 umull 命令のサイクル数についての考察

μNaCl で採用されたプリミティブのうち，Curve25519 は 255bit の楕円曲線の演算を行うため 256bit 乗算を，Poly1305 ではメッセージを 16byte に区切り，1 を 1byte 付与した 17byte のチャンクに対し演算を行うため 136bit 乗算が必要である．加えて，256bit 乗算をカラツバ法を用いて計算するため，128bit 乗算が必要となる．これら多倍長乗算の実装は AVR NaCl と同様に，128bit と 136bit 乗算をすべてのループを展開した筆算方式を用いて，256bit 乗算をコードサイズ削減のためカラツバ法を用いて実装する．

ARM Cortex-M3 マイコンの umull 命令は，32bit レジスタ同士の乗算結果（64bit）を，32bit レジスタ 2 つに格納する命令である．この命令を多倍長乗算に用いることで，8bit AVR に比べ大幅な性能向上が期待できる．しかし，ハードの仕様より，この命令は入力値により早期終了される場合があり，実行サイクルが 3 から 5 サイクルで変化する [12, pp.3-5]．この実行サイクルの変化はタイミング攻撃に悪用される恐れがあるため，悪用されないように実行サイクルを固定したい．

そこで本実装では，乗算をすべて 16bit ずつ，1 サイクルで終了する mul 命令で行う．これにより多倍長乗算の実行サイクルは増えるが，実行サイクルは入力によらず一定となり，安全性を高めることができる．

3.2 SHA2-512 の最適化についての方針

μNaCl の採用するハッシュ関数 SHA2-512 は，AVR NaCl の実装をそのまま ARM Cortex-M3 上に実装しただけでは，1.5 倍程度の性能しか得られない．原因は AVR NaCl では SHA2-512 で多用される 64bit 長 n-bit 右シフト演算と 64bit 右ローテーション演算にある．AVR マイコンは，シフトとローテーション命令が 1bit ずつしか行えないという制限がある．そのため，64bit シフトとローテーション演算も 1bit ずつ処理するコードとなっていた．これに対して，ARM Cortex-M3 マイコンにはそのような制限はないため，64bit シフトとローテーション演算を最適化することで，SHA2-512 の性能向上が期待できる．

一般に用いられている 64bit 右ローテーションをシフト演算と論理和演算を用いて行うアルゴリズムを，Algorithm 1 に示す．また，実際に 32bit CPU 向けに実装したものを Algorithm 2 に示す．なお，64bit 値 a の上位 32bit を a_1，下位 32bit を a_0 で表す．

Algorithm 1 64bit architecture implemented 64bit-length n-bit right rotate

Ensure: $0 \leq n \leq 64$
$a_{left} \leftarrow a \ll (64 - n)$
$a_{right} \leftarrow a \gg n$
$a \leftarrow a_{left} \mid a_{right}$
return a

Algorithm 2 によって，64bit 右ローテーションが高速化される．更に，この処理から条件分岐処理を取り除くことで，更なる安全性の確保，高速化，コードサイズの削減を行う．

Algorithm 2 の IF∼THEN∼ELSE 文内で行っている処理は，64bit 値 a の上位 32bit と下位 32bit の交換である．32bit アーキテクチャ上で $32 \leq n$ の場合は，64bit 値 a の上位 32bit を下位 32bit に，下位 32bit を上位 32bit に置き換えた後，$(32 - n)$bit の右ローテーションを行わなければならない．この処理を行うことで，以降のローテーション処理を $n < 32$ の場合と共通化している．

Algorithm 2 32bit architecture implemented 64bit-length n-bit right rotate

Ensure: $0 \leq n \leq 64$
  if $n < 32$ **then**
    $a0 \leftarrow a_0$
    $a1 \leftarrow a_1$
  else
    $a0 \leftarrow a_1$
    $a1 \leftarrow a_0$
    $n \leftarrow n - 32$
  end if
  $a0_{left} \leftarrow a0 \ll (32 - n)$
  $a0_{right} \leftarrow a0 \gg n$
  $a1_{left} \leftarrow a1 \ll (32 - n)$
  $a1_{right} \leftarrow a1 \gg n$
  $a_0 \leftarrow a1_{left} \mid a0_{right}$
  $a_1 \leftarrow a0_{left} \mid a1_{right}$
  return a

交換処理は，上位 32bit を表す変数 $a1$ と下位 32bit を表す変数 $a0$ を作り，a_0，a_1 を代入する順番を入れ替えることで行っている．この a_0，a_1 を a_i で表し，インデックス i の値を値 n からビット演算で計算すると，Algorithm 3 のように条件分岐処理を削除できる．

Algorithm 3 64bit-length n-bit right rotate

Ensure: $0 \leq n \leq 64$
  $index \leftarrow (n \gg 5)\ \&\ 1$
  $a0 \leftarrow a_{index}$
  $index \leftarrow index \oplus 1$
  $a1 \leftarrow a_{index}$
  $n \leftarrow n\ \&$ 0x1f
  $a0_{left} \leftarrow a0 \ll (32 - n)$
  $a0_{right} \leftarrow a0 \gg n$
  $a1_{left} \leftarrow a1 \ll (32 - n)$
  $a1_{right} \leftarrow a1 \gg n$
  $a_0 \leftarrow a1_{left} \mid a0_{right}$
  $a_1 \leftarrow a0_{left} \mid a1_{right}$
  return a

同様の手法を用いることで，64bit 右シフトも最適化が可能である．

3.3 乱数生成についての考察

AVR NaCl の論文で提案された乱数生成の手法は，外部の乱数生成装置を用いるか，RC オシレータのジッターを乱数生成手法 [13] に用いるものであった．前者の外部装置を用いる方法は安全な乱数を得られるが，部品が増えるためコストが増加する．これは部品コストの増加を避けるべき組み込み製品では好ましくない．後者の RC オシレータのジッターを用いる方法は，追加の部品はほとんど必要ないが，1 秒間に 1byte しか生成できない．加えて電源に対する Frequency injection 攻撃 [14] が発見されている．

そこで，本実装の擬似乱数生成機能は，起動直後の SRAM の値をエントロピー源とする手法 [15] と，OpenBSD などで使われる乱数シードを ChaCha20 でかく乱して擬似乱数を生成するプログラム “arc4random” を参考に，SRAM の初期値をエントロピー源として抽出し，Salsa20 でかく乱して擬似乱数を生成する．Salsa20 は μNaCl に既に実装されているため，コードサイズの増加を抑えられる．

3.3.1 乱数生成手順

本実装の乱数生成の手順を以下に示す．

1. 電源を投入する
2. SRAM の初期値を 12,800bit 取得する
3. 取得した値を SHA2-512 に通し，乱数シードを作る
4. 乱数シードを Salsa20 の内部状態に設定する
5. Salsa20 で 20 ラウンド計算し，256bit の乱数を出力する

3.4 性能評価

3.4.1 乱数の安全性考察

μNaCl で唯一乱数が利用されるのは，32byte の鍵生成時である．この乱数に求められるエントロピーについて考える．乱数シードのエントロピーは NIST SP800-90[16] にて，共通鍵の長さの 1.5 倍以上が推奨されている．よって乱数シードのエントロピーは，384bit 以上が推奨される．電源投入後の SRAM に含まれるエントロピーは，1bit あたり 3%である [17]．

よって，電源投入後に SRAM 上から 12,800bit（1,600byte）の値を取得し，SHA2-512 でエントロピー抽出を行ったものを乱数シードとした．この方式で生成した 20,000bit の乱数は，FIPS 140-2 のテストに合格した．

3.4.2 ベンチマーク結果

AVR NaCl と本実装のベンチマーク結果を表 3 に示す．なお，比較対象には AVR NaCl（avrnacl-20140813, fast）を Arduino Mega 2560（ATMEGA2560）上で実行したベンチマーク結果を用いる．

本実装の性能を AVR NaCl と比較した結果は概ね 3 倍から 4 倍となり，コードサイズは約半分に減少した．この結果は Thumb2 命令セットにより多くの命令長を 16bit に抑えられたことと，AVR マイコンでは 8bit ずつ行っていた多倍長演算処理を，32bit ARM マイコンでは約 4 倍の効率で実装できたことの結果である．

4 おわりに

本論文では 8bit AVR マイコンへ実装された暗号ライブラリ AVR NaCl を基に，ARM Cortex-M3 用 暗号ライブラリを実装した．そして，本実装が 8bit AVR マイコン実装に比べ，約半分のコードサイズと，4 倍弱の高速化を達成していることを確認した．

本実装は今後の課題として，更なる速度の改善が挙げられる．例えば，Düll らによって 2015 年 4 月に発表された Curve25519 の AVR, MSP430，ARM Cortex-M0 マイコンでの高速実装 [18] では，256bit 乗算への 3 段階カラツバ法の適用，2 乗演算の高速実装等を行うことにより，ARM Cortex-M0 において 3589850 クロックサイクルという結果を導いている．この結果は本実装に比べ 1.7 倍程度高速であり，改良の参考とする予定である．

SRAM に含まれるエントロピーの割合は，ST マイクロ社の ARM マイコンである STM32 マイコンについての結果 [17] を利用したが，他社の ARM マイコンで同じ結果が得られるか，未調査である．ARM マイコンは ST マイクロ社や NXP 社以外に，FreeScale 社やルネサス社など，様々なメーカーが製造している為，これら全てのメーカが作る ARM マイコンの SRAM について，どれだけのエントロピーが含まれるか，調査の必要がある．

今回実装したライブラリを mbed で利用できるよう移植を行えば，より多くの人が手軽にこのライブラリの恩恵を受けられるようになり，mbed を利用したプロダクト全体のセキュリティが向上できるであろう．

参考文献

[1] ARM. ARM IoT Overview, May 2014.

[2] Daniel J Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *Progress in Cryptology–LATINCRYPT 2012*, pp. 159–176. Springer, 2012.

[3] Michael Hutter and Peter Schwabe. NaCl on 8-Bit AVR Microcontrollers. In *AFRICACRYPT*, pp. 156–172. Springer, 2013.

[4] Daniel J Bernstein. Cryptography in nacl, 2009.

[5] Daniel J Bernstein. Curve25519: new Diffie-Hellman speed records. In *Public Key Cryptography-PKC 2006*, pp. 207–228. Springer, 2006.

[6] Daniel J Bernstein. The Salsa20 family of stream ciphers. In *New stream cipher designs*, pp. 84–97. Springer, 2008.

[7] Daniel J Bernstein. The Poly1305-AES message authentication code. In *Fast Software Encryption*, pp. 32–49. Springer, 2005.

[8] FIPS PUB. Secure Hash Standard (SHS), 2012.

[9] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *Computer Security–ESORICS 2011*, pp. 355–371. Springer, 2011.

[10] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *Topics in Cryptology–CT-RSA 2007*, pp. 225–242. Springer, 2006.

[11] Inc. Software in the Public Interest. Debian security advisory, DSA-1571-1 openssl—predictable random number generator, 2008.

[12] Cortex-M3 テクニカルリファレンス マニュアル リビジョン r2p1.

[13] Josef Hlavác, Róbert Lórencz, and Martin Hadácek. True random number generation on an Atmel AVR microcontroller. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, Vol. 2, pp. V2–493. IEEE, 2010.

[14] Simona Buchovecka and Josef Hlavác. Frequency injection attack on a random number generator. In *DDECS'13*, pp. 128–130, 2013.

[15] Anthony Van Herrewege and Ingrid Verbauwhede. Software Only, Extremely Compact, Keccak-based Secure PRNG on ARM Cortex-M. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pp. 111:1–111:6, New York, NY, USA, 2014. ACM.

[16] John Barker, Elaine; Kelsey. NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, January 2012.

[17] Anthony Van Herrewege, Vincent van der Leest, André Schaller, Stefan Katzenbeisser, and Ingrid Verbauwhede. Secure PRNG Seeding on Commercial Off-the-shelf Microcontrollers. In *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, TrustED '13, pp. 55–64, New York, NY, USA, 2013. ACM.

[18] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography*, pp. 1–22, 2015.

表 3: μNaCl ベンチマーク結果

API	Message bytes	Execute Cycle		Gain
		AVR	ARM	AVR / ARM
crypto_auth_hmacsha512256	1024	6367104	1720925	370%
crypto_auth_hmacsha512256_verify	1024	6367512	1721264	370%
crypto_box_curve25519xsalsa20poly1305_keypair		23239860	6256950	371%
crypto_box_curve25519xsalsa20poly1305_beforenm		22853464	6250195	366%
crypto_box_curve25519xsalsa20poly1305	1056	23342860	6407177	364%
crypto_box_curve25519xsalsa20poly1305_open	1056	23061840	6316794	365%
crypto_box_curve25519xsalsa20poly1305_afternm	1056	489830	156987	312%
crypto_secretbox_xsalsa20poly1305	1056	489812	156752	315%
crypto_secretbox_xsalsa20poly1305_open	1056	208536	66543	313%
crypto_hash_sha512	1024	4773346	1277880	374%
crypto_dh_curve25519_keypair		23239836	6276088	370%
crypto_dh_curve25519		22836606	6264332	365%
crypto_stream_salsa20	1024	266092	80972	329%
crypto_stream_salsa20_xor	1024	281300	91339	308%
crypto_stream_xsalsa20	1024	282728	87163	324%
crypto_stream_xsalsa20_xor	1024	298210	97520	306%
Primitives				
crypto_core_hsalsa20		17019	5122	332%
crypto_core_salsa20		16930	5086	333%
crypto_hashblocks_sha512	1024	4239316	1145076	370%
crypto_onetimeauth_poly1305	1024	173767	56100	310%
crypto_onetimeauth_poly1305_verify	1024	174005	56285	309%
crypto_scalarmult_curve25519_base		22836588	6245458	366%
crypto_scalarmult_curve25519		22836588	6245448	366%
crypto_sign_ed25519_keypair		21913629	5534610	396%
crypto_sign_ed25519	1024	22691528	5897934	385%
crypto_sign_ed25519_open	1088	37562656	9577407	392%
crypto_verify_16		383	231	166%
crypto_verify_32		553	375	147%

NaCl implementation	Code size(bytes)		Difference
	26924	15240	11684