Renesas マイコン RL78, RH850 上における Chaskey の性能評価

渡辺 大†　　森田 伸義†

† 株式会社日立製作所
244-0817 神奈川県横浜市戸塚区吉田町 292
dai.watanabe.td@hitachi.com

あらまし Mouha らが提案した Chaskey は、32 ビットマイコンにおける性能を重視したメッセージ認証子であり、提案論文の中で ARM Cortex-M プロセッサにおける性能評価を行っている。本稿では、ルネサステクノロジのマイコン RL78 および RH850 上で Chaskey の性能評価を行った結果について紹介する。

The performance evaluation of Chaskey on Renesas micro-controllers RL78 and RH850

Dai Watanabe†　　Nobuyoshi Morita†

†Yokohama Research Laboratory, Hitachi, Ltd.
292 Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, JAPAN
dai.watanabe.td@hitachi.com

Abstract The message authentication code Chaskey proposed by Mouha *et al.* is designed to be suitable for 32-bit processors and the performance analysis on ARM Cortex-M is given in the proposal paper. In this paper, we implement Chaskey and evaluate its performances on Renesas micro-controllers RL78 and RH850.

1　はじめに

近年、クラウドに代表される情報システムの集約と、スマートフォンの普及に伴い、携帯端末を介した情報の収集、分析が一般的になっている。このような背景をもとに、これまで独自のネットワークでシステムを構築してきた産業・インフラなどの制御システムをインターネットに接続し、システム間でリソース活用の最適化を図るなど、システム管理の効率化を目指す動きが出てきている。これらの「モノのインターネット (Internet of Things)」の中には、米 General Electric が提案する Industurial Internet [7] や独政府が主導する Industrie 4.0 [12] など、水平分業化した産業システム群を 1 つのインフラと捉え、効率的な利活用を目指すような、産業構造の大規模な変革を標榜する動きも見られる。また、自動車業界に目を向ければ、運転自動化に向けて各種車載システムを高度情報化する動きが活発化している。

しかしながら、これらのシステムはネットワークが外部と切り離されていることを前提に設計・運営されてきた経緯から、情報系システムに比べて情報セキュリティ対策が大幅に遅れているのが現状である。その一方で、Stuxnet [18] のように、産業系をターゲットとした攻撃は増加傾向にある。自動車についても、Kohno らが現状の車載システムの脆弱性を明らかにした [3] ことを契機としてセキュリティ対策の必要性に関する議論が活発に行われるようになっている。

そこで、基本的な暗号機能、特に改ざん検知機能を実装することで、仮想的にネットワークの安全性を保証する対策が考えられる。しかし、産業系と自動車ではコスト意識が高く、暗号機能の導入に伴うネットワーク負荷の増加や端末での処理負荷の増加を忌避する傾向が見られる。そこで、導入に伴うコスト増ができるだけ小さい暗号技術 (軽量暗号) を提供することで、IoT システムにおけるセキュリティ対策の導入を加速することが求められている。Chaskey [17] は、Mouha らが 32 ビットマイコン向けに開発した MAC アルゴリズムであり、ARM Cortex-M 上で優れた性能を発揮すると主張しており、IoT システムに適している可能性がある。

本稿では、ルネサスエレクトロニクスの 32 ビットマイコン RH850 と 8 ビットマイコン RL78 上において、AES と Chaskey のアセンブリ実装および性能比較を行う。実装評価の結果、RH850 では AES の 5.6 倍、RL78 では AES の 2.8 倍の処理性能であることを確認した。また、車載システムに接続された ECU の処理性能、ネットワークの帯域などのパラメータを加味し、AES と Chaskey がシステムにどの程度の負荷をかけるかについて検討を行った結果について報告する。

2 Chaskey

2.1 アルゴリズム仕様

一般的な MAC の構成法としては、ブロック暗号ベースの CMAC [14]、ハッシュ関数ベースの HMAC [15] がある。これに対し、Chaskey の大域構造は、非線形置換をベースにしている (図 1 参照)。

Chaskey の大域構造は、非線形置換からブロック暗号を構成する Even-Mansour 構成法 [5] と、前記の CMAC を組み合わせたものとみなすことができる。通常のブロック暗号 (鍵付き非線形置換) では、秘密鍵から繰り返し処理 (ラウンド処理) で使うラウンド鍵を生成する鍵スケジュール処理が存在するのに対し、Even-Mansour ブロック暗号では鍵スケジュール処理を含まない。このことから、RAM の消費が少ないと期待される。

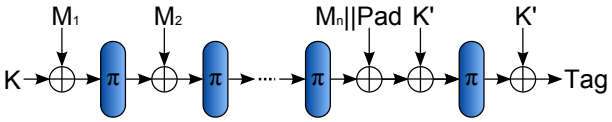

図 1: Mode of operations of Chaskey

また、Chaskey の非線形置換は、入出力長 128 ビットであり、32 ビット単位の算術加算、巡回シフト、排他的論理輪のみで構成されている (図 2)。これらの演算のみで非線形処理を構成する方法を一般に ARX (Addition, Rotation, Xoring) 構造と呼ばれ、次世代ハッシュ関数標準を選定する SHA3 コンペティション [16] で finaists に選ばれた Skein [6] や、Python の標準ハッシュ関数に採用された SipHash [1] などが知られている。いずれのアルゴリズムも、ソフトウェア実装において AES に代表されるテーブル参照型のアルゴリズムと比較して高速な処理が可能である。

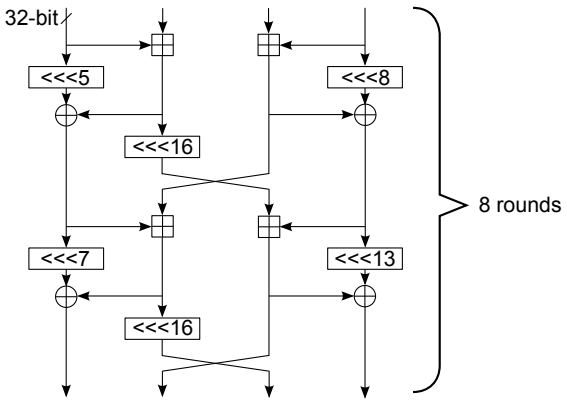

図 2: The round function of Chaskey

2.2 既存の実装評価結果

Chaskey の提案論文 [17] では、ARM Cortex-M0, -M3/4 プロセッサ上において Chaskey と AES の性能を比較評価している。表 1 は、[17] からの抜粋である。この実装結果から、Chaskey

表 1: Performance comparison of AES and Chaskey on Cortex-M [17]

CPU	アルゴリズム	ROM(B)	処理速度 (cycles/B)
Cortex-M0	AES-128-CMAC	10,816	136.5
		8,988	140.0
	Chaskey	1,308	18.3
		414	16.9
Cortex-M3/4	AES-128-CMAC	11,512	105.0
		8,740	89.4
	Chaskey	908	10.6
		402	11.2

のコードサイズは AES の 1/10 であり、処理速度は 7〜10 倍であることがわかる。

3 32 ビットマイコン RH850 上での実装評価

3.1 RH850 のアーキテクチャ

RH850 はルネサステクノロジ (以下、ルネサス) のマイコンである [20]。RH850 はバス幅 32 ビット、レジスタ長 32 ビットの CPU であり、5〜7 段のパイプラインを持つ。

3.1.1 レジスタ

RH850 は 32 個の汎用レジスタを持つ。ただし、そのうち 8 個は用途が定まっているため、一般には 24 個のレジスタを使用して演算を行う。

3.1.2 命令セット

RH850 の命令セットは 32 ビット単位でデータ転送、算術加算、排他的論理和、巡回シフトを行う命令を含み、いずれの命令も 1 サイクルで実行できる。

3.1.3 メモリ

RH850 のメモリサイズは、ROM が 256KB〜2MB、RAM が 32〜192KB である。命令キャッシュを搭載している型もあるが、データキャッシュは搭載していない。

3.2 実装方針

RH850 の命令セットを使えば、Chaskey の仕様を素直にアセンブリ実装することが可能であるため、特殊な最適化は不要である。また、RH850 はメモリが 1MB 以上あるため、RAM/ROM のサイズに関する制約は低いと考えられるので、RH850 については、省メモリ実装は行わない。

AES については、[13] に記載されている、一般的な最適化方法を用いている。RH850 はメモリアクセスのレイテンシが大きいので、命令の順序を適宜入れ替えることで、命令の実行密度を高めている。

3.3 性能評価

表 2: Implementation results on RH850

アルゴリズム	言語	ROM (B)	処理速度 (cycle/B)
AES	C	5,746	46.1
	Assembly	5,142	41.3
Chaskey	C	584	10.4
	Assembly	1,156	7.4

表 2 は、Chaskey と AES の処理性能を比較したものである。この結果から、RH850 上では Chaskey は AES の約 5.6 倍の処理速度であることが確認できた。また、ROM の使用量はいずれのアルゴリズムも数 KB であり、実装上の制約とならない程度であると考える。

4 8 ビットマイコン RL78 上での実装評価

4.1 RL78 のアーキテクチャ

RL78 はルネサスのマイコンである [19]。RL78 はハーバードアーキテクチャで、3 段のパイプラインを持つため、命令やデータの呼び出し／書き出しは比較的ボトルネックになりにくい[1]。

4.1.1 レジスタ

RL78 は、8 つの汎用レジスタ a，x，b，c，d，e，h，l を持ち、これらは 4 つの 16 ビットレジスタ ax，bc，de，hl として利用することができる。ax レジスタはアキュムレータとして利用させており、計算の入力の 1 つ、および出力は ax に格納する必要がある。また、hl レジスタのみがアドレスポインタとして利用可能である。この他にも、命令によって利用可能なレジスタが制限されている。

4.1.2 命令セット

RL78 の大半の命令は 8 ビット単位であり、データ転送や算術加算など一部の命令のみ 16 ビット処理をサポートしている。特に、16 ビットのキャリー付き算術加算命令、排他的論理和命令が無いため、Chaskey の実装においては、算術加算と排他的論理和はほぼ 8 ビット演算で構成することになる。

[1] R8C はノイマンアーキテクチャであり、8 ビットのデータバス 1 本で命令とデータを転送する。また、RL78 に比べて命令長が長く、データ転送がボトルネックとなっていた

表 3: Instructions of RL78 used for the implementations of Chaskey

8 ビット命令	mov，add，addc，xor，inc dec
16 ビット命令	movw，xchw，addw，shlw shrw
制御命令	cmp，skz，br

4.1.3 メモリ

RL78 はいくつかのグループに分けられているが、たとえば車載用で比較的低スペックに属する RL78/F12 では、RAM サイズが 0.5〜4K バイト、ROM サイズが 8〜64K バイトである。

4.2 実装方針

RL78 の命令セットの特徴として、RAM 上に配置されたデータへのアクセスのコストが小さいことが挙げられる。算術加算 add を例にとると、レジスタ reg とアキュムレータ ax 上のデータの加算 add reg，ax は命令調が 2 バイトで実行に 1 サイクルを要するのに対し、メモリ mem と ax 上のデータの加算 add mem，ax では命令長が 3 バイト、実行には 1 サイクルを要する。すなわち、二項演算を行う場合、(ax 上にない) データがレジスタ上にあるか、それともメモリ上にあるか、の違いは命令長が 1 バイト異なるのみで、処理速度には影響しない。単項演算でも、inc，dec はメモリ上のデータを直接処理可能である。

これとは別に、RL78 は 8 ビットの汎用レジスタを 8 個持つだけなので、Chaskey の内部状態 (16 バイト) を格納することができない。したがって、レジスタ上に効率的にデータを配置して処理を高速化することは、基本的に難しいと考えられる。

以上の 2 点から RL78 上での実装では、すべてのデータはメモリ上に配置したままとし、処理対処のデータのみをアキュムレータ上に移動する方針とした。高速実装では、関数をすべて展開

した。一般的にはループ処理について unroll することが高速化に繋がるが、Chaskey のアルゴリズムでは、ラウンド依存のパラメータが無く、unroll により削減できるのはループの終了条件の判定のみであるため、我々の実装では unroll を行わないこととした。表 4 は算術加算、巡回シフト、排他的論理和の実装をまとめたものである。

4.3 性能評価

本節では、4.2 節で述べた方針に従って Chaskey を実装した結果について報告する。第 5 節で詳述するが、RL78 は CAN に接続される ECU であり、処理するパケットは基本的に 8 バイトである。そこで、Chaskey の完全な仕様を実装するのではなく、1 ブロックの暗号化に相当する処理のみを実装した。

また、比較対象としては AES を選択した。AES の RL78 上での性能については、[10, 11] で詳しく調べられているため、これらの結果を引用した。比較を厳密なものとするため、関数インターフェースは [10, 11] に合わせた。

表 5 は、Chaskey と AES の処理性能を比較したものである。この結果から、高速実装においては、Chaskey は AES に比べて約 2.8 倍の処理速度を達成していることが確認できた。省メモリ (ROM) 実装では、処理速度の差は 2 倍程度に縮まっているが、これは、AES に比べて Chaskey の実装バリエーションが少ないことに由来する。

5 車載ネットワークにおける Chaskey の有用性

本節では、第 4 節の実装結果をもとに、車載ネットワークにおける暗号処理の負荷について検討する。

5.1 処理されるデータ量の限界

まず、ネットワーク帯域の限界について検討する。CAN の帯域は 1 Mbps であり、芳賀らの報告 [8] によれば、実際には CAN の帯域の 50%程度を利用するに留めるようにマージンが設定されている。したがって、帯域は最大で 500 Kbps と想定すれば良い。

一方、ECU の通信性能の観点では、それぞれのチップが持つ CAN および Ethernet のポート数を見れば良い。CAN については、RL78/F13, F14, F1A のいずれも 1 チャネルを持ち、1 チャネルあたり送信ポートが 4 個、受信ポートが 16 個ある。CAN のパケットは最大 108 ビットであること (図 3 参照)、また、多くの ECU は 10 ミリ秒を周期として処理を行っていることから、RL78 が実際に処理できるデータ量の上限は 216 Kbps と見積もられる。本稿では、後者を ECU の実質的なデータ処理能力の限界と想定して、MAC の計算負荷を検討する。

5.2 MAC の計算頻度

前述のように、CAN のパケットは最大 108 ビットであり、1 パケットあたり 64 ビットのペイロードを持つ (図 3 参照)。一般に、ペイロードの領域すべてが既に利用されていることが多い [8] ことが知られており、各々のパケットには、MAC 値を格納するための空き領域が無い。つまり、MAC 値は別送する必要がある [2]。

1 つの CAN パケットに対して 1 回の MAC 計算を行うと、通信量は 2 倍に増加する。したがって、通信量の増加を抑えるためには、MAC の計算／送信頻度を下げる方が良い。その一方で、MAC 値の送信頻度が下がると、リアルタイムで改ざんを検知できなくなるため、攻撃に対する対処が遅れ、車載システムの安全性が低下する。また、偶発的にパケット損失が発生した場合も MAC の検証に失敗するため、MAC の送信頻度を決定するには、ネットワーク品質も考慮に入れる必要がある。実際のシステム設計では、これらのトレードオフを理解した上で、適切にパラメータを設定することが望ましい。

本報告では、CAN パケット 1〜8 個ごとに MAC 値を送信するケースを想定して見積もり

[2] 実際には、同期ずれを抑止するために同期パケットを送信することも必要だが、今回の検討では想定から外している

表 4: Implementations of Addtion, Rotation, and Xoring on RL78

Addition	Rotation	Xoring
`movw ax, [hl+tgt0]`	`movw ax, [hl+tgt0]`	`movw ax, [hl+tgt0]`
`addw ax, [hl+src0]`	`movw bc, ax`	`xor a, [hl+src1]`
`movw [hl+tgt0], ax`	`shlw ax, left`	`xch a, x`
`movw ax, [hl+tgt2]`	`movw de, ax`	`xor a, [hl+src0]`
`xch a, x`	`movw ax, [hl+tgt2]`	`xch a, x`
`addc a, [hl+src2]`	`shrw ax, right`	`movw [hl+tgt0], ax`
`xch a, x`	`addw ax, de`	`movw ax, [hl+tgt2]`
`addc a, [hl+src3]`	`movw [hl+tgt0], ax`	`xor a, [hl+src3]`
`movw [hl+tgt2], ax`	`movw ax, [hl+tgt2]`	`xch a, x`
	`xchw ax, bc`	`xor a, [hl+src2]`
	`shrw ax, right`	`xch a, x`
	`shlw bc, left`	`movw [hl+tgt2], ax`
	`addw ax, bc`	
	`movw [hl+tgt2], ax`	

表 5: Implementation results on RL78

アルゴリズム	実装方針	RAM (B)	ROM (B)	処理速度 (cycle/B)
AES [10]	高速実装	60	1,021	3,855
	省 ROM 実装	78	486	7,288
Chaskey	高速実装	37	437	1,357
	省 ROM 実装	47	264	3,712

を行う。

5.3 RL78 における CAN データの処理負荷

RL78 の動作周波数は 32 MHz であるため、1 個パケットの処理に割ける時間は最大で 16,000 サイクルである。図 4 は、AES, Chaskey のそれぞれの処理速度から、ECU の負荷を算出したものである。

このグラフから、Chaskey であれば、CPU 占有率 10%以下で MAC の計算処理が可能であることがわかる。

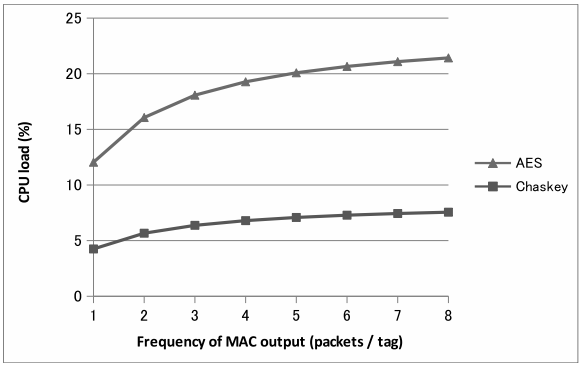

図 4: CPU loads of crypto-processing on RL78, the transmission limitation of the chip is considered

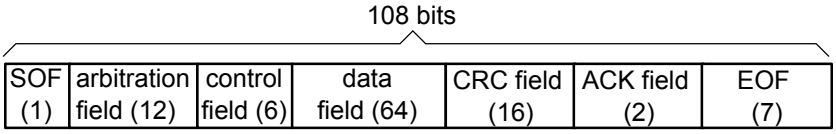

図 3: CAN packet format [9]

6 まとめ

本報告では、Mouha らが提案した MAC アルゴリズム Chaskey について、ルネサスの車載向け 32 ビットマイコン RH850 と 8 ビットマイコン RL78 上で実装最適化について報告した。RH850 上では 8 cycle/B、RL78 上では 84.5 cycle/B を達成した。これらの性能は、それぞれ AES の 5.6 倍、2.8 倍の速度である。また、上記の処理速度と、マイコンのネットワーク処理性能をベースに、RL78 上における CPU の暗号処理負荷を見積もった。

参考文献

[1] J. P. Aumasson, "SipHash: a fast short-input PRF", `https://131002.net/siphash/`.

[2] Robert Bosch GmbH, "CAN with Flexible Data-Rate", Specification Version 1.0, April 17th, 2012, `http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf`.

[3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces", USENIX Security, August 10-12, 2011.

[4] O. Dunkelman, N. Keller, A. Shamir, "Minimalism in Cryptography: The Even-Mansour Scheme Revisited", *Advances in Cryptology, Eurocrypt 2012*, Lecture Notes in Computer Science, vol. 7237, pp. 336–354, Springer, 2012.

[5] Shimon Even and Yishay Mansour, "A Construction of a Cipher From a Single Pseudorandom Permutation", *Journal of Cryptology*, 1991, pp. 151–161, Springer-Verlag.

[6] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker, "The Skein Hash Function Family", `http://www.skein-hash.info/`.

[7] General Electrics, "Industrial Internet: Pushing the Boundaries of Minds and Machines", November 26, 2012. `www.ge.com/sites/default/files/Industrial_Internet.pdf`. [2015 年 6 月 18 日閲覧].

[8] 芳賀 智之, 岸川 剛, 氏家 良浩, 松島 秀樹, 田邉 正人, 北村 嘉彦, 安齋 潤, "車載ネットワークを保護するセキュリティ ECU の提案: 導入インパクトを抑えた CAN 保護手法のコンセプトとその評価", 暗号と情報セキュリティシンポジウム, SCIS 2015, 3C2-3, 2015.

[9] 五十嵐資朗, 佐藤正幸, 玉城礼二, 「CAN 入門講座〜組み込みマイコンで学ぶ CAN プロトコルとプログラミング」, 電波通信社, 2006.

[10] 村上ユミコ, 松井充, "RL78 におけるブロック暗号のソフトウェア実装性能評価", 暗号と情報セキュリティシンポジウム, SCIS 2013, 1B1-4, 2013.

[11] Mitsuru Matsui and Yumiko Murakami, “Minimalism of Software Implementation – Extensive performance Analysis of Symmetric Primitives on the RL78 Microcontroller”, *Fast Software Encryption, FSE 2013*, Lecture Notes in Computer Science, vol. 8424, pp. 393–409, 2013.

[12] National Academy of Science and Engineering, “Securing the future of German manufacturing industry: Recommendations for implementing the strategic initiative INDUSTRIE 4.0”, Final report of the Industrie 4.0 Working Group, April 2013. http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Material_fuer_Sonderseiten/Industrie_4.0/Final_report__Industrie_4.0_accessible.pdf [2015 年 6 月 18 日閲覧].

[13] National Institute of Standards and Technology, “Advanced Encryption Standard”, Federal Information Processing Standards Publication 197, November 26, 2001.

[14] National Institute of Standards and Technology, “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication”, NIST Special Publication 800-38B, May, 2005.

[15] National Institute of Standards and Technology, “The Keyed-Hash Message Authentication Code (HMAC)”, Federal Information Processing Standards Publication, FIPS 198-1, July, 2008.

[16] National Institute of Standards and Technology, “SHA-3 Competition (2007-2012)”, http://csrc.nist.gov/groups/ST/hash/sha-3/.

[17] Nickey Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede, “Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers”, *Selected Areas in Cryptography, SAC 2014*, Lecture Notes in Computer Science, vol. 8781, pp. 306–323, Springer, 2014.

[18] 小熊 信孝, “Stuxnet–制御システムを狙った初のマルウエア–”, http://www.jpcert.or.jp/ics/2011/20110210-oguma.pdf, [2015 年 6 月 18 日閲覧].

[19] ルネサスエレクトロニクス, 「RL78 ファミリ ユーザーズマニュアル ソフトウェア編」, Rev.2.20, 2014.11.

[20] Renesas, RH850 ファミリ (車載用), http://japan.renesas.com/products/mpumcu/rh850/index.jsp