6-6　IoT 端末向けの認証プロトコルの開発と実装評価

森山大輔

2015 年頃から、Internet of Things（IoT）による新たな情報供出が将来的に行われるようになると言われている。本研究では、省リソースの IoT 機器向けに、正しい端末が通信を行っているかを検証する暗号学的な認証プロトコルの開発を行い、FPGA を介したソフトウェア・ハードウェアの実装を行うまでの流れと評価を行った。

1　まえがき

2015 年頃から、ICT 技術においてクラウドコンピューティングの次に来る時代の流れとして様々な産業から IoT が話題になってきた。セキュリティアーキテクチャ研究室では、クラウドから末端の計算資源に制約がある端末までを包括的にサイバー攻撃から守るアーキテクチャを設計することを 2011 年度からの第 3 期中長期計画の目標のひとつとし、本著者は様々な海外の研究者と共同研究を行いつつ、IoT 端末において重要な役割を担う RFID タグを代表例とした認証プロトコルや、様々な応用プロトコルの研究開発を行ってきた。IoT 端末に組み込まれる機器として細分化した場合、代表的なものはセンサと RFID タグの 2 つに分類される。センサにおける主な役割は、温度・湿度・震度などセンサ自身により収集した電子データを（センサ間などの通信を通して）サーバに伝送するものであり、セキュリティの観点としてはデータ保護やセンサの位置の秘匿などが主に研究されている。一方、RFID タグの場合は様々なモノに付着され、モノの ID を正しく認証することが主な目的とされており、なりすまし防止やプライバシー保護がセキュリティとして重要視されている。

本研究では、これらの IoT 端末が非常に強力な安全性やプライバシー保護を必要とした場合の認証プロトコルを開発し、かつプロトコルをどのような手順を踏むことで実装が行われるべきかについて考察した。既存研究はいくつか挙げられるものの、安全性やプライバシーに関して厳密な安全性証明を与えてあり、そして十分に安全性やプライバシーが示された認証プロトコルに対しての実装はこれまで行われていなかった [1]。本研究では、Google/Columbia 大学の Moti Yung 氏及び Virginia 工科大学の Patrick Schaumont 准教授の協力の下で理論研究と実装研究を組み合わせ、電子回路の製造ばらつきを指紋のように扱い、固有の値を導く物理的複製困難関数（PUF）を利用した証明可能安全な認証プロトコルの開発と、プロトコルに必要な構成要素をどのように評価・分析し実装するべきかという手順について議論し、最終的に FPGA によるソフトウェア及びハードウェアを実装して評価を行った。

2　PUF ベースの認証プロトコルの構築

2.1　構成要素

本稿においては、以下の暗号学的な関数を利用する。

- 乱数生成器 : TRNG は真の乱数列を出力する。
- 物理的複製困難関数（PUF）: 関数 $f: K \times D \to R$ は物理特性 $x \in K$ とメッセージ $y \in D$ から出力 $z \in R$ を求める。物理特性は特に IC 回路における製造ばらつきを利用し、個々の端末において他の端末とは相関性がない出力を行う関数を構成するものとする [2]。
- 共通鍵暗号 : SKE :=（SKE.Enc, SKE.Dec）は共通鍵暗号方式を表し、SKE.Enc は秘密鍵 sk と平文 m から暗号文 c を求め、SKE.Dec は秘密鍵 sk と暗号部 c から平文 m を復元する。
- 擬似ランダム関数 : PRF,PRF': $K' \times D' \to R'$ は秘密鍵 $sk \in K'$ とメッセージ $m \in D'$ から、乱数と識別不可能なビット列を出力する。
- ファジィ抽出機 : FE :=（FE.Gen, FE.Rec）はファジィ抽出機を表すものとする。FE.Gen は変数 z を入力とし乱数 r とヘルパーデータ hd を出力する。FE.Rec は z との距離が近い値 z' を hd と共に入力した場合、r を復元する。もし、距離が d 以下であり z の最小エントロピーが h 以上である場合 (d, h)- ファジィ抽出機は hd が公開されても r が真性乱数と統計的に識別不可能であることを保証する。ファジィ抽出機の構成には、エラー訂正符号と乱数抽出機を組み合わせたものを利用

することが多い [3]。

2.2　認証の安全性モデル

本研究では、IoT の環境としてひとつのサーバが複数の機器（総数を num 個とする）と通信するものとし、悪意のある攻撃者が端末のプライバシーを脅かすことがないプライバシーのレベルや、中間者攻撃に対する耐性を要求するものとする。特に、通信内容については盗聴されている可能性や通信内容が改変されていることを念頭に、認証結果や端末内部の不揮発性メモリの中身についても（物理攻撃を通して）得られることを想定する。その状況下において、いかなる確率的多項式時間攻撃者に対しても、中間者攻撃により通信路上のデータを改ざんしたとしてもサーバや端末が認証を受理しない場合、認証プロトコルは安全性があるという。また、端末から漏洩させた情報や通信路上のデータからを解析したとしても、どの端末から情報が出力されたかを識別することができない場合、認証プロトコルはプライバシーを満たすという。

2.3　安全かつプライバシー保護型認証プロトコル

図 1 に著者が提案した認証プロトコルの流れを示す。プロトコルの構成には PUF を利用しており、端末固有の関数であることからサーバは事前に一度端末に対して入力 y_1 を送り、その返答 z_1 を受け取っておく（この処理はオフラインで安全に行われるものとする）。また、暗号化に利用する鍵 sk をサーバに送り端末は sk と y_1 を不揮発性メモリに保存する。その後、認証プロトコルにおいて端末は PUF とファジィ抽出機を利用して乱数 r_1 を生成して擬似ランダム関数 PRF を用いたチャレンジレスポンスの相互認証を行いつつ、ヘルパーデータ hd は sk で暗号化する。また、別の入力による PUF の出力を XOR で暗号化するための乱数やメッセージ全体に対してのメッセージ認証子としての役割を持つ鍵、安全性に配慮するために更新する鍵についても PRF によって出力を行う。具体的な特徴としては以下が挙げられる。

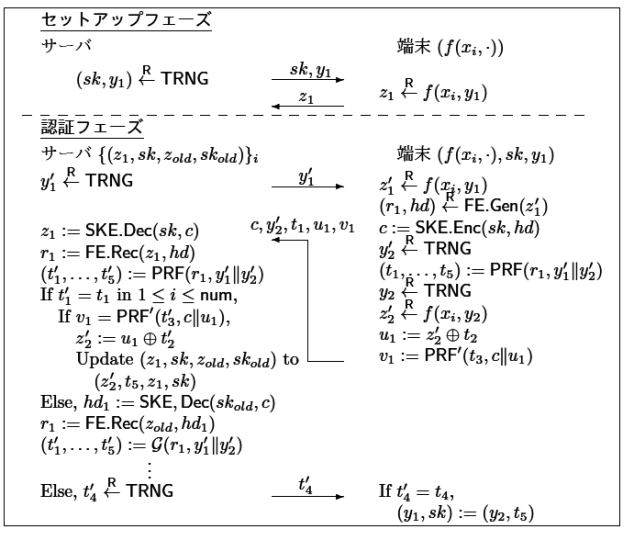

図 1　認証プロトコルの流れ

- PUF による鍵導出：
  セットアップにおいてサーバは PUF の出力 z_1 を保存する。認証フェーズにおいて端末は物理特性 x_1 を用い $z_1' \leftarrow f(x_i, y_1)$ を求めるが、z_1 とは異なるためファジィ抽出器を用いヘルパーデータを $(r_1, hd) \leftarrow \mathrm{FE.Gen}(z_1')$ として求める。ヘルパーデータは暗号化して送られ、サーバは復号した後 $r_1 := \mathrm{FE.Rec}(z_1, hd)$ を求めるため、両者で同じ（ランダムな）鍵を導出することができる。つまり、PUF を利用するため端末は r_1 を不揮発性メモリに保存する必要が無い。そのため、たとえ悪意のある攻撃者が不揮発性メモリの中身をのぞくことができたとしても、本提案プロトコルは安全性を保つことが可能となる。
- 相互認証と安全なメッセージ送信：
  相互に鍵 r_1 を導出したら、擬似ランダム関数を用い、ビット列 $(t_1, \cdots, t_5)$ を求める。(t_1, t_4) は相互認証に用いられ、t_2 は PUF の出力の XOR 暗号化に用いられる。t_3 は全体のメッセージの検証を行う値 v_1 を生成するための鍵として用いられ、t_5 は更新用の鍵となる。
- 全数探索：
  端末はプロトコルの通信においてプライバシー保護のため固有の値や ID を出力しない。その代わりに、サーバはデータベースの中のインデックス $i \in \{1, \cdots, num\}$ を全数探索し、対応する鍵を見つける。全数探索は非効率ではあるもののプライバシーに配慮するためには必要不可欠であり、一般的に RFID 認証においては広く知られている方法である。

3　構成要素の実装

2 で述べたプロトコルについては、理論面での安全性が証明可能であることを示すことができるが（詳細は [4]）、プロトコルをどのように実装すべきかについて検討を行う必要がある。ここでは、安全性レベルを 128 bit として評価した際の実装に必要なそれぞれの変数の長さや処理方法について議論する。

3.1　アーキテクチャ

実装環境としては、暗号研究において標準的に利用されている国産 FPGA ボード SASEBO-GII を利用し

た。SASEBO-GII には 2 Mbit の SRAM（ISSI 社 の IS61 LP6432 A）及び 16 MBit のフラッシュメモリ（ATMEL AT45 DB161 D）が載せられている。SRAM は 32 bit 出力の 64 K のメモリであり、フラッシュメモリは SPI 接続にて FPGA と通信可能である。

3.2　SRAM PUF のデザイン

PUF に関しては様々な研究が行われているが、本研究ではコスト効率が一番高い SRAM PUF を利用することとした。SRAM PUF は SRAM への電源投入時の（能動的な書き込みを行っていない）不定状態を観測することにより、それぞれのチップ固有の値を導くことができることから PUF の性質を満たすことが期待されている。PUF を評価するにあたり必要なのは、エントロピーの推定と出力ごとのノイズである。

3.2.1　エントロピー

PUF の出力自体は真性乱数ではないため、どの程度の乱数性が含まれているかをエントロピー推定を行うことで求める。90 台の SASEBO-GII から 2 Mbit のデータを 11 回観測し、全体 990 × 2 Mbit のデータを分析した。

シャノンエントロピー率については、データを n bit に分割し確率 b_i でそれぞれの値を出力するとした場合、$\sum_{i=0}^{n} -b_i \log(b_i)/n \times 100$ により求めることができる。n の値には依存せず、端末によるばらつきを考慮すると 34–46 % となった。シャノンエントロピーは平均的な情報量をとらえたものであるが、最悪状態を考える最小エントロピー率を求める場合 $n \times \min_i -b_i \log(b_i) \times 100$ という計算を行う必要がある。ビット単位で考えるとシャノンエントロピー率とあまり差が生じなかったが、バイト単位（$n = 8$）で計算した場合、最小エントロピー率は 5–15 % という分析結果になった。この原因は、多くの SRAM において 0 xAA が観測されていたことに起因する。この問題を回避するため、32 bit のデータごとに 2 つの 16 bit データに分割し、それらを XOR することにより偏りを中和させることにした。その結果、それぞれの端末ごとに求めた最小エントロピー率のうち最小のものでも 26 % という結果が得られた。

3.2.2　ノイズ

もうひとつ、実際に PUF の利用に影響を及ぼすのはノイズである。例えば SRAM PUF の場合、2 回観測したとしても同じデータが得られるわけではなくノイズが載るため、若干異なる値が出力される。そのためにはエラー訂正符号をファジィ抽出器内部で活用するが、そのためのパラメータを特定するためにノイズがどの程度生じるかを推定する必要がある。

上記のエントロピー処理によってデータを XOR しているためノイズがその分増えるが、平均的には 64 bit に対して 6.6 bit のノイズが載ることが分かったため、ノイズ量は 10 % と見積もった。なお、異なる PUF とのハミング距離を計測した場合、平均で 64 bit 中 31.9 bit となったため、異なる SRAM が同一のものだと判断されることはない。

3.2.3　乱数生成器としての利用

一般的に、暗号プロトコルにおいては（暗号学的に安全な）乱数が利用されるが、小さな IoT 機器においては乱数の生成元にコストがかかる場合がある。一方、我々の場合 SRAM PUF のノイズが 10 % あることから、XOR 処理を何度か行うことで乱数を生成することが可能であり、実際に 8 つのデータを XOR した場合に NIST 乱数テスト検定 [5] に通ることが確認された。この場合、1,024 bit の SRAM の生データから 128 bit の乱数を生成することができる。我々のプロトコルにおいては 652 bit の乱数が必要となるため、乱数生成を行うために SRAM に必要な生データは合計 5,216 bit である。

3.3　共通鍵暗号と擬似ランダム関数

提案プロトコルは、IoT 機器向けの認証であるため、共通鍵暗号としては軽量ブロック暗号として知られている SIMON[6] を選んだ。SIMON は他の軽量暗号よりも優れているという評価が多く、また、複数の安全性レベルをサポートしている。さらに、擬似ランダム

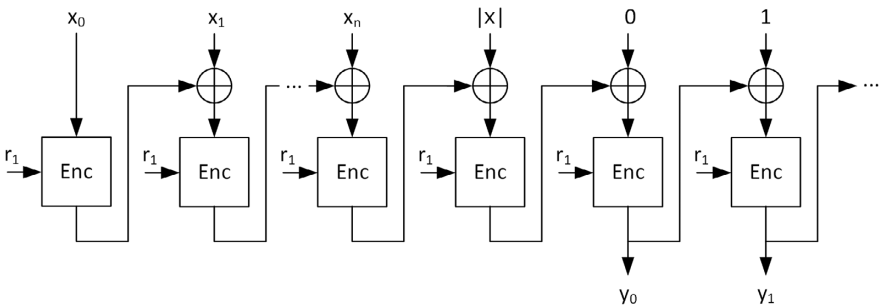

図 2　共通鍵暗号を利用した擬似ランダム関数

表 1　提案プロトコルのデータ長と鍵長

カテゴリー	目的	変数	64-bit security	128-bit security
セットアップ	入力アドレス	y_1	12	12
	PUF の出力	z_1	252	504
	記憶する鍵	sk, sk'	64	128
認証フェーズ	PUF の出力	z_1', z_2'	252	504
	ナンス	y_1', y_2'	64	128
	乱数（ファジィ抽出器）	δ, rnd	128	256
	PRF の秘密鍵	r_1	64	128
	ヘルパーデータ	hd (includes rnd)	632	1,264
	暗号文	c	640	1,280
	PUF の入力	y_2	12	12
	相互認証	t_1, t_4	64	128
	XOR 演算要素	t_2	252	504
	PRF′ と MAC の鍵	t_3, s_1	64	128
	更新される鍵	t_5	128	256
通信データ	最初のメッセージ（サーバ）	y_1'	64	128
	2 番目のメッセージ（端末）	(c, y_2', t_1, u_1, s_1)	1,084	2,168
	3 番目のメッセージ（サーバ）	t_4'	64	128
メモリ	状態情報（不揮発性メモリ）	(sk, sk', y_1)	140	268
	PUF のための SRAM 領域		504	1,008
	RNG のための SRAM 領域		2,656	5,312

関数に関しては、SIMON における暗号化関数 Enc を CBC モードで利用し、入力メッセージ $(x_0, \cdots, x_n)$ をブロック暗号の平文ブロックとした後、入力サイズ $|x|$ 及びカウンタを入力し、必要な出力長となる文までカウンタを増加させていく。図 2 に実装を行った擬似乱数生成器の仕組みを記載する。

3.4　ファジィ抽出器

3.4.1　エラー訂正符号

PUF のデータを事後処理する方法はいくつか検討されているが、本研究では BCH 符号を用いた code-offset というメカニズムを利用する。(**BCH.Gen, BCH. Dec**) を BCH 符号アルゴリズムとした場合、

- Encode(a): $\delta \leftarrow \mathrm{TRNG} \in \{0,1\}^{k_1}, cw := \mathrm{BCH.Gen}(\delta) \in \{0,1\}^{n_1}, hd := a \oplus cw$
- Decode(a', hd): $cw' := a' \oplus hd, cw := \mathrm{BCH.Dec}(cw'), a := cw \oplus hd$

という手順でデータの復元を行う。入力となる a は乱数元 δ による XOR で暗号化しているため、hd がもし漏れたとしても a が直接求まることはない。

実際に PUF のデータに (n_1, k_1, d_1) -BCH 符号を適応する場合、PUF の出力 z_1 は複数の $n_1$bit に分割されるため、総当りとしても $2^{k_1 \cdot |z_1|/n_1}$ の計算が必要となる。この値を 128 bit よりも大きくすることが必要となる。前述の SRAM PUF の解析結果より最小エントロピー率は 26 % であるため、504 bit のデータを 8 つに分割し (63, 16, 23) -BCH 符号を適応すると、全体として $504 \times 0.26 > 128$ bit 分の乱数が含まれることになる。

一方、(63, 16, 23) -BCH 符号は $(23-11)/63 \times 100 = 17.5\%$ のエラー訂正を行うことができるものの、SRAM PUF においてそれぞれのビットが 10 % のノイズを含むことを考慮すると 63 bit 中エラーが 12 bit 以上になる確率は 2.36 % であり、全 8 ブロックがすべて正しく復号される確率は $(1-0.0236)^8 \times 100 = 82.6\%$ に留まる。そのため、インターリーブ符号の原理を利用して元データを行列のように配置して転換させたデータにもう一度同じパラメータの code-offset によるエラー訂正を行うこととした。結果的に、ヘルパーデータは 2 倍になるものの、8 ブロックが正しくエラー訂正される確率は $1 - 1.92 \times 10^{-6}$ にまで改良することができた。

3.4.2　乱数抽出器

乱数抽出器は、一様でないビット列（本研究の場合は PUF から生成されたデータ）から乱数を抽出するアルゴリズムである。本研究では、上記にて提案した共通鍵暗号を基にした擬似ランダム関数を乱数抽出器とした。乱数抽出器は基本的には確率的アルゴリズムであるため、乱数成分を必要とする。既存研究から、乱数の長さは安全性レベルの 2 倍以上の長さが必要なため、128 bit 安全性である場合は 256 bit の乱数が必要である。

最終的に、本章での解析結果によるプロトコル上の各データや変数の長さをまとめたものは表 1 となる。

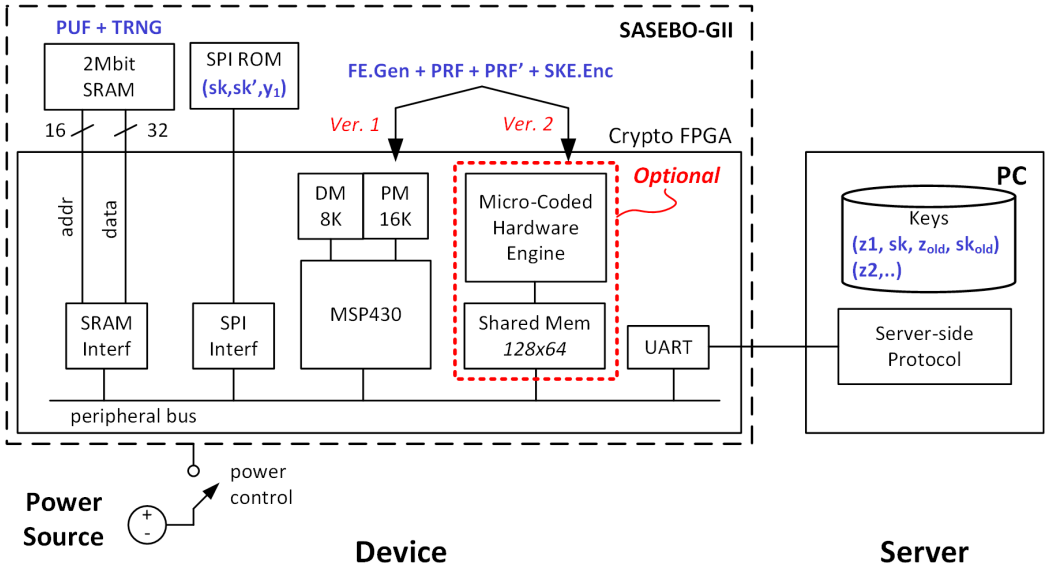

図 3　サーバと端末の実装評価アーキテクチャ

4　アーキテクチャデザイン

提案プロトコルにおけるサーバと端末のシステムアーキテクチャを図 3 に示す。実装では、これらはそれぞれ PC と SASEBO GII にエミュレートされ、特に端末側については SASEBO GII に載っている FPGA 内部で MSP430 というマイクロコントローラをソフトコア実装させる。そして、場合によりハードウェア実装による効率性を比較するためハードウェアエンジンとして FPGA に直接記述した暗号処理を動作させる。SASEBO GII 上には SRAM や不揮発性メモリ (EEPROM) が載っており、適宜プロトコルの実行に沿ってこれらにアクセスを行う。ソフトコア実装された MSP430 にはプログラムメモリとデータメモリがあり、プログラムメモリに暗号プロトコルの処理が書き込まれ、それぞれの変数の値がデータメモリに書き込まれる。サーバ側は秘密鍵と端末から得られた PUF の出力値に関する情報がデータベースとして保管されており、本研究における実装ではサーバと端末間の通信は USB ケーブルによるシリアル通信としている。

ハードウェアエンジンには SIMON による暗号化、SIMON を利用した擬似ランダム関数の計算、BCH 符号の計算を記述しており、ハードウェアエンジンと MSP430 とは共通メモリが仲介し、データの受け渡しを行う。ハードウェアエンジンを利用する際は、MSP430 のデータメモリから必要な情報を共通メモリにコピーし、ハードウェアレベルでの上記処理を行った後に結果を再度共有メモリに書き込み、MSP430 に読み込ませることで通信を行っている。この通信量の分オーバーヘッドがかかるが、次章にてソフトウェア実装のみよりも高速に動作することを示す。

表 2　MSP430 のメモリフットプリント (バイト)

カテゴリー	64-bit MSP430	128-bit MSP430	128-bit MSP430 + HW
HW 抽出	1,022	1,022	1,398
通信	496	644	628
SIMON	1,604	2,440	0
BCH 符号化	1,214	1,214	0
PUF + FE	562	646	590
乱数生成器	396	456	396
プロトコル	1,568	1,682	1,908
テキスト量	6,862	8,104	4,920
変数	424	656	656
定数	197	197	73
データ量	621	853	729

5　実装評価

本章では、我々のプロトコルを実装する際に端末に必要な実装コストと計算量について実装評価する。本研究では、MSP430 によるソフトウェア実装による 64 bit と 128 bit 安全性の場合と、ハードウェアエンジンを利用した場合の 128 bit 安全性の計 3 種類の実装を行った。

5.1　実装コスト

表 2 は 3 種類の実装におけるメモリ消費量を表しており、MSP430 のオブジェクトコードやデータメモリの消費量を含めたものである。MSP430 のためのコンパイラは GNU gcc version 4.6.3 を最適化レベル 2 で利用している。MSP430 のメモリ量は 8KB であるため、3 種類すべてにおいて実装可能である。

表 3　提案プロトコルの計算量 (サイクル)

プロトコル処理	実装ターゲット	64-bit	128-bit	128-bit with HW
sk, sk', y_1 読込	ROM 読込	31,356	61,646	61,646
$y_2' \xleftarrow{R}$ TRNG, $y_2 \xleftarrow{R}$ TRNG	SRAM 乱数生成	11,552	23,341	22,981
$z_1' \xleftarrow{R} f(x_i, y_1), z_2' \xleftarrow{R} f(x_i, y_2)$	SRAM PUF	4,384	9,082	8,741
$(r_1, hd) \xleftarrow{R}$ FE.Gen(z_1')	BCH 符号 乱数抽出器	268,820 28,691	485,094 205,080	18,597
$(t_1, \ldots, t_5) :=$ PRF$(r_1, y_1' \| y_2')$	擬似ランダム関数 PRF	44,355	299,724	
$c :=$ SKE.Enc(sk, hd)	暗号化	39,583	252,829	
$v_1 :=$ PRF$'(t_3, c \| u_1)$	擬似ランダム関数 PRF$'$	57,601	394,126	
全体		486,343	1,730,922	111,965
y_2, t_5 書込	ROM 書込	76,290	128,829	128,849

5.2　計算量

提案プロトコルに対しての 3 種類の実装の計算量をシステムクロック単位で比較したものを表 3 に記す。IoT 機器が省リソース端末であることを踏まえ、MSP430 は 1.846 MHz で動作させた。実装においては、ハードウェアエンジンを利用した場合大幅に計算量が削減されている。また、表におけるデータにはハードウェアエンジンへのデータ転送時間が含まれており、実際の暗号処理にかかる計算量は 4,486 クロックである。

6　まとめ

本研究では、IoT 機器向けの匿名認証プロトコルを通じて理論からソフトウェア・ハードウェア実装までの道のりとその評価について解説した。プロトコルのすべての構成要素がひとつの実装として具体化した評価を行うことは、将来的な実際の端末での実現可能性を検証する非常に重要な事柄である。本成果の目的は、どのようにプロトコルを具体的に実装するかについて焦点を置いており、幾つかの改良の余地が残されている。例えば、アーキテクチャのレベルとして計算量・実装コスト・消費電力などの項目に応じた最適化は可能であると思われる。また、別の PUF や軽量な共通鍵暗号方式、エラー訂正符号による実装についても異なる結果が得られるであろう。その結果、導出された最適な認証プロトコルが将来的な民間企業が製造する様々な IoT 機器に実装され、利用者のセキュリティ・プライバシに配慮した ICT 社会が実現されることが望まれる。

【参考文献】

1 Delvaux,J., Gu,D., Peeters,R., and Verbauwhede,I., “A survey on lightweight entity authentication with strong PUFs,” IACR Cryptology ePrint Archive 2014/977, 2014.
2 Maes,R., “Physically Unclonable Functions - Constructions, Properties and Applications,” Springer, 2013.
3 Dodis,Y., Ostrovsky,R., Reyzin,L., and Smith,A., “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” SIAM J. Comput. 38 (1) , pp.97–139 , 2008.
4 Aysu,A., Gulcan,E., Moriyama,D., Schaumont,P., and Yung,M., “End-to-end design of a PUF-based privacy preserving authentication protocol,” In: CHES 2015. LNCS, vol.9293, pp.556–576. Springer, Heidelberg. Full version is available at IACR Cryptology ePrint Archive 2015/937, 2015.
5 Rukhin,A., Soto,J., Nechvatal,J., Smid,M., Barker,E., Leigh,S., Levenson,M., Vangel,M., Banks,D., Heckert,A., Dray,J., and Vo,S., “A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications,” Special Publication 800-22 Revision 1A, April, 2010.
6 Beaulieu,R., Shors,D., Smith,J., Treatman-Clark,S., Weeks,B., and Wingers, L., “The SIMON and SPECK families of lightweight block ciphers,” IACR Cryptology ePrint Archive 2013/404, 2013.

森山大輔　（もりやま　だいすけ）
元ネットワークセキュリティ研究所
セキュリティアーキテクチャ研究室
研究員
博士 (情報学)
暗号プロトコル