マイクロコントローラのスリープ状態に着目した消費電力にもとづく悪意のある機能の発現検知

長谷川 健人[1,a)]　柳澤 政生[1]　戸川 望[1]

概要：近年 IoT デバイスが普及しており，多くの製品にマイクロコントローラが内蔵されている．マイクロコントローラは内蔵ソフトウェアが書き換え可能であり，悪意のある機能を挿入される危険性が指摘されている．本稿ではマイクロコントローラを対象として，消費電力にもとづき悪意のある機能の発現を検知する手法を提案する．マイクロコントローラを搭載したデバイスとして，省電力化のためスリープ状態を有するものを対象とする．まず，マイクロコントローラを搭載したデバイスの消費電力を測定し，スリープ状態と通常状態の区間を教師なし学習で識別する．次に，通常状態の区間に着目してその継続時間とその間の消費エネルギーを算出し，特徴量を抽出する．抽出した特徴量にもとづき，外れ値検知アルゴリズムを適用して悪意のある機能の発現を検知する．実験では，マイクロコントローラ上に悪意のある機能を実装し，実際に悪意のある機能の発現の検出に成功した．

1. はじめに

近年，高機能なハードウェアデバイスが我々の身の回りで普及している．これらのデバイスには，専用のアプリケーションを実現するため ASIC (Application Specific Integradet Circuit) や FPGA (Field Programmable Gate Array)，マイクロコントローラなどの SoC (System on Chip) が利用されている．ハードウェアデバイスの普及に伴い，それらに対するセキュリティを考える必要がある．特に近年の IoT (Internet of Things) 化により，ハードウェアデバイスはネットワークに接続されるようになっており，ハードウェアデバイスに対するセキュリティリスクはますます増大している．

IoT デバイスは，大きく分けて (i) アプリケーション層，(ii) ネットワーク層，(iii) ハードウェア層から構成される [1]．アプリケーション層では，IoT デバイスにおける主要な機能を実現する．ネットワーク層では，IoT デバイスにおいてネットワーク通信を実現する．ハードウェア層では，IoT デバイスにおけるセンサなどのハードウェアを利用する．IoT デバイスに対する脅威を考えるとき，それぞれの層に対するセキュリティが検討される．このうち，アプリケーション層とネットワーク層に対するセキュリティはこれまでにも様々な研究がなされている一方で，ハードウェア層に対するセキュリティ対策はアプリケーション層とネットワーク層に対する対策と比べ不十分である．本稿では，ハードウェア層に対するセキュリティに着目する．

ハードウェアデバイスは様々な部品から構成される．そのうち，プロセッサが演算の中心的な役割を担う．プロセッサにはハードウェアデバイスの用途により ASIC，FPGA，あるいはマイクロコントローラなどの SoC が利用される．攻撃者にとってこれらの SoC に対し悪意のある機能を埋め込むことでその存在を隠すことができるため，これらの SoC は標的となり得る．ASIC や FPGA に対する悪意のある機能は「ハードウェアトロイ」と呼ばれ，近年その攻撃手法や防御手法が研究されている [2]．ASIC は特定のアプリケーション専用の回路であり，FPGA は回路設計情報を SoC にダウンロードすることで任意の回路を実装できるものであり，両者はともに回路を利用して機能を実現する点で共通する．この点で，パス遅延やゲートの構成など回路の特性を利用したハードウェアトロイ検出手法が提案されている [3, 4]．一方，マイクロコントローラは CPU (Central Processing Unit) やメモリを 1 つの SoC に内蔵したものであり，その機能はソフトウェアで実現される．そのため，マイクロコントローラのソフトウェアに挿入された悪意のある機能を，従来のハードウェアトロイ検出手法を用いて検出することはできない．また，マイクロコントローラは一般的なコンピュータと比較して計算資源の制約が厳しいことから OS (Operating System) を搭載しないことが多く，特定のプラットフォームで動作することを前提とする従来のマルウェア検出手法も適用することが難しい．この点において，マイクロコントローラに対する

[1] 早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻
Dept. of Computer Science and Communications Engineering, Waseda University
[a)] kento.hasegawa@togawa.cs.waseda.ac.jp

悪意のある機能を検知する技術は，現状では議論が不十分である．ASIC や FPGA ではハードウェアに特化した専門知識が必要となる一方で，マイクロコントローラは機能をソフトウェアで実現でき，C 言語などのプログラミング言語を用いて機能を実装できる．マイクロンコントローラのアプリケーションは通常書き換え可能であるため，ネットワーク経由でソフトウェアを更新できる反面，同様にネットワーク経由で悪意のある機能を挿入される危険性も考えられる．以上より，マイクロコントローラを対象とした悪意のある機能を検知する技術の検討は，近年の大きな課題の一つである．

本稿では，マイクロコントローラの消費電力にもとづき，マイクロコントローラのソフトウェアに挿入された悪意のある機能の発現を検知する手法を提案する．消費電力にもとづくハードウェアトロイ検出手法はこれまでにも研究されている [5]．しかし，これらのハードウェアトロイ検出手法はハードウェアトロイが挿入されていないことが保証されている「ゴールデン回路」を必要とする．ゴールデン回路はそれ自体を保証するのが困難であるため，現実的ではない．本稿では，センサロガーなどの小規模で省電力動作が必要な IoT デバイスを対象とする．これらの IoT デバイスは通常の機能を間欠的に動作させ，それ以外の時間はスリープ状態として電力の消費を抑える．このように，小規模で省電力な IoT デバイスでは**通常モード**と**スリープモード**を交互に繰り返しながら動作する．本稿ではこの点に着目し，IoT デバイスの通常モードの継続時間とその間の消費エネルギーを測定することで，悪意のある機能を検知することを目的とする．

本稿の貢献を以下に示す．

(1) 消費電力にもとづき，教師なし学習により IoT デバイスが通常モードかスリープモードかを自動的に識別する．
(2) 通常モードの継続時間とその間の消費エネルギーを測定することで，外れ値検知アルゴリズムを用いて悪意のある動作の発現を検知する．

本稿は以下のように構成される．2 章で，本稿で対象とする IoT デバイスの脅威モデルを明らかにする．3 章で，マイクロコントローラの通常モードとスリープモードの違いに着目した消費電力にもとづく悪意のある機能の発現検知手法を提案する．4 章で，検知対象のアプリケーションを実装したマイクロコントローラに対し提案手法を適用した実験結果を示す．5 章で本稿をまとめる．

2. 脅威モデル

本章では，本稿で対象とするマイクロコントローラに対する脅威モデルを明らかにする．

2.1 マイクロコントローラに対する悪意のある機能

マイクロコントローラは 1 つの SoC に CPU やメモリ，その他周辺機能を搭載したものである．マイクロコントローラに書き込まれるプログラムは C 言語などのプログラミング言語で記述することができる．記述されたプログラムをコンパイルされた実行バイナリは，シリアル通信など経由してマイクロコントローラに書き込まれる．近年ではインターネットに接続可能なマイクロコントローラも発表されており，これらの高機能なマイクロコントローラではネットワークを経由してプログラムを更新することも可能となる．このように，IoT デバイスでは内蔵されるプログラムの書き換えが容易なため，攻撃者にとっても悪意のある機能を挿入するのが容易である．これらの脅威に対抗するため，プログラムの書き換えを不可能にしたり，プログラム更新時にプログラムの正当性を評価する機能を搭載したマイクロコントローラが存在する一方で，こうした機能はマイクロコントローラの利便性や開発効率を損なうこともあり，安価な IoT デバイスではしばしば無視されることがある．

マイクロコントローラに挿入された悪意のある機能は，情報漏洩や機能の無効化を引き起こす．悪意のある機能は一般に，1) 常に動作し続けるものと，2) 特定の条件を満たした場合にだけ動作するものに分けられる．1) 常に動作し続けるものは，マイクロコントローラが動作している間は常に悪意のある機能が動作する．これにより消費電力を増加させ，IoT デバイスの耐久性や信頼性を低下させる．2) 特定の条件を満たした場合にだけ動作するものは，動作時刻やセンサ等の入力値などが特定の条件を満たした場合にだけ動作する．これにより，出荷時のテスト段階では悪意のある機能が隠蔽され，通常利用している間の任意のタイミングでだけ悪意のある機能が動作し，その時点の内部情報流出などを引き起こす．本稿では，2) 特定の条件を満たした場合にだけ動作するような悪意のある機能を対象とする．

特定の条件を満たした場合にだけ動作するような悪意のある機能の場合，短期間の動作テストでは悪意のある機能を検知することは難しい．本稿で提案する手法は，工場内で実際の製品利用を想定した長期間の動作テストにおいて適用されることを想定する．

2.2 マイクロコントローラの動作モード

センサロガーなどの小規模な IoT デバイスはマイクロコントローラを搭載する．これらの IoT デバイスに搭載されるマイクロコントローラでは，電力の消費を抑えるため一般に通常モードとスリープモードの 2 つの動作モードをもつ．

- **通常モード**: 通常モードでは，マイクロコントローラは通常の処理を実行する．例えばセンサロガーの場合，マイクロコントローラはセンサから情報を取得し，その情報を内部メモリに記憶または外部に接続されたホストコンピュータへセンサ情報を送信する．通常処理では，マイクロコントローラは数十から数百 mW 程度

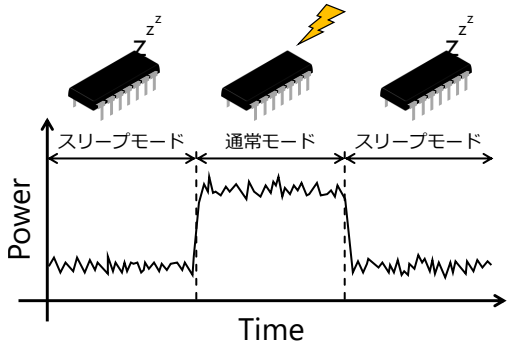

図 1　通常モードとスリープモードのときの消費電力の例.

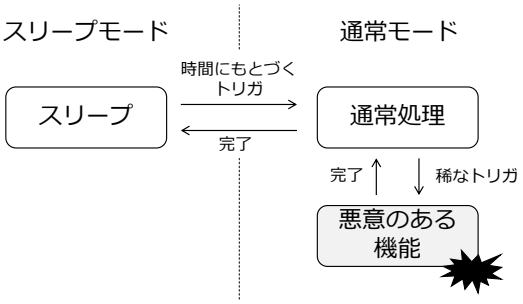

図 2　想定する，スリープモードと通常モードにおける動作の様子.

の電力を消費する.

- **スリープモード**: スリープモードでは，マイクロコントローラの主要な機能を停止する．次回スリープモードから復帰するために最低限必要な機能以外の機能を無効化することで，電力消費を抑える．スリープモード中のマイクロコントローラの消費電力は，通常モード中の消費電力と比較して非常に小さい.

図 1 に，通常モードとスリープモードにおけるマイクロコントローラの消費電力の概形を示す．消費電力は時間によってわずかに変動するが，通常モードとスリープモードの間の消費電力は明らかに異なる．本稿では，この消費電力の差に着目する.

図 2 に，本稿で想定するマイクロコントローラの動作モードを示す．スリープモードでは，マイクロコントローラでは主要な機能が停止しているため，何もしない．動作モードでは，通常の機能が動作する．ここで，悪意のある動作がマイクロコントローラに挿入された場合，通常モードにおいて，ごく稀に発生するトリガ条件にもとづき悪意のある機能が動作すると仮定する．この場合，通常モード時の消費電力において，悪意のある機能が動作していない場合に比べ，通常モードの継続時間やその間の消費エネルギーに変化が生じる．本稿では，この変化に着目し，悪意のある機能の発現を検知する.

3. 消費電力にもとづく悪意のある機能の発現検知

本章では，消費電力解析にもとづく，マイクロコントローラに挿入された悪意のある機能の発現検知手法を提案

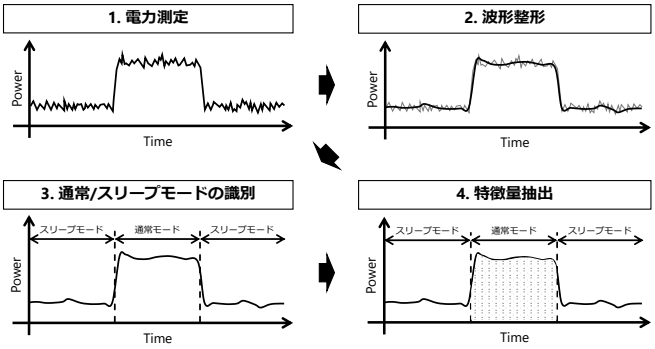

図 3　提案手法の処理の流れ.

する．提案手法は，以下の 5 つの処理から構成される.

(1) **電力測定**: 測定機器を用いて対象となる IoT デバイスの消費電力を計測する.

(2) **波形整形**: 測定時のノイズを軽減するため，消費電力の測定波形を平滑化する.

(3) **通常/スリープモードの識別**: 教師なし学習を用いて，消費電力波形を通常モードの区間とスリープモードの区間に識別する.

(4) **特徴量抽出**: 通常モードと識別された区間に対し，その継続時間とその間の消費エネルギーを算出し，特徴量として抽出する.

(5) **外れ値検知**: 抽出された特徴量に対し外れ値検知アルゴリズムを適用し，異常な区間を検出する．検出された区間では，悪意のある機能が発現したと識別される.

以降の節では，それぞれの詳細な手順を示す.

3.1 Step 1: 電力測定

対象の IoT デバイスから消費電力を計測する．本稿では一般的なオシロスコープを用いることとする．オシロスコープを用いて消費電力 P を計測する場合，通常は電圧 V と電流 I を別々に測定する．これらの結果にもとづき，$P = V \times I$ を計算することで消費電力 P を得る．オシロスコープを用いて電流を測定する場合，a) シャント抵抗を用いる方法と，b) 電流プローブを用いる方法が考えられる.

a) シャント抵抗を用いる場合では，シャント抵抗と呼ばれる抵抗値が高精度に定められた非常に小さい抵抗器を利用する．電流を測定する信号線に対しシャント抵抗を挿入し，その両端の電圧をオシロスコープで計測する．シャント抵抗の抵抗値は高精度に定められているため，計測の結果得られた電圧値をシャント抵抗の抵抗値で割ることで，電流値を得られる．この手法では対象となる回路を改変するため，被測定回路への影響が大きいばかりでなく，多くのデバイスを対象として測定する場合は工数が多いため現実的でない．一方で，b) 電流プローブを用いる場合，電流を測定する信号線の周囲を電流プローブで囲うことで電流を測定する．電流プローブにはホール素子と呼ばれるセンサが内蔵されており，電流が発生する磁界を検知して電圧に変換することで電流を計測する．電流プローブを用いた

計測は対象の回路を直接改変する必要がないため，対象回路への影響を最小限に抑えることができる．本稿では，電流プローブを用いる手法で電流を計測する．

3.2 Step 2: 波形整形

Step 1 で電流プローブを用いて計測された波形はノイズを含むため，平滑化する．本稿では単純に平滑化するため，移動平均を利用する．時刻 n における時系列データ $x[n]$ に対し，N 次の移動平均を取る場合，移動平均により平滑化されたデータ $y[n]$ は，$y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i]$ で示される．本稿では $N = 5$ として 5 次の移動平均をとる．

3.3 Step 3: 通常/スリープモードの識別

Step 2 で整形した波形に対し，通常モードの区間かスリープモードの区間かを識別する．消費電力波形に対し通常モードかスリープモードかを識別するとき，i) 閾値を設定する方法と，ii) 教師なし学習を適用する方法が考えられる．i) 閾値を設定する方法では，IoT デバイスが通常モードかスリープモードかを識別するための閾値を設定し，消費電力が閾値を上回った場合を通常モード，下回った場合をスリープモードとして識別する．閾値を設定する方法はごく単純に実装できる反面，予め閾値を厳密に設定する必要がある．IoT デバイスの種類がますます増加している昨今において，それぞれの IoT デバイスに対し個別に閾値を設定することは現実的でない．ii) 教師なし学習を適用する方法では，予め閾値を設定することなくアルゴリズムで自動的に通常モードかスリープモードかを識別する．教師なし学習のうち，本稿ではクラスタリングアルゴリズムに着目する．k 平均法 [6] は，教師なし学習によるクラスタリングアルゴリズムの一つである．k 平均法では，与えられたデータを予め設定したクラスタ数 k 個に分割したときそれぞれのクラスタの重心からそのクラスタに属するデータまでの距離の総和を最小化する．k 個のクラスタ s_i において μ_i をクラスタ s_i における重心 (平均値) として，各クラスタの重心の集合は $C = \{\mu_1, \mu_2, \ldots, \mu_k\}$ としたとき，N 個のデータ $x_1, x_2, \ldots, x_N$ に対し，k 平均法は $\sum_{i=1}^{N} \min_{\mu_j \in C}(\|x_i - \mu_j\|^2)$ を最小化するようデータをクラスタに分類する．本稿では，平滑化した消費電力波形に対し，k 平均法を適用して通常モードとスリープモードの区間を識別する．識別には通常モードとスリープモードの 2 種類に分割するため，本稿では k 平均法の k を 2 と設定する．

3.4 Step 4: 特徴量抽出

本節以降，Step 3 で識別された通常モード区間の消費電力に注目する．本節では，通常モード区間の消費電力波形から，通常モードの継続期間とその間の累積消費電力，すなわち消費エネルギーを特徴量として抽出する．

本稿では，悪意のある機能はトリガ条件にもとづき作動するものとする．悪意のある機能は，A) 通常の機能とは別の動作をする悪意のある機能を追加するような種類と，B) 通常の機能を無効化するような種類に大別される．A) 悪意のある機能を追加するような種類では，通常モードの区間において悪意のある動作を実行するため通常よりも多くの処理時間が必要となる．この種類の悪意のある機能として，内部情報の流出が挙げられる．B) 通常の機能を無効化するような種類では，通常モードの区間において悪意のある動作により通常の処理が無効化されるため，通常よりも処理時間が短くなる．この種類の悪意のある機能として，DoS (Denial-of-Service) が挙げられる．いずれの場合においても，悪意のある機能が動作した場合，通常モードの継続時間やその間の消費電力に何らかの影響を及ぼすと考えられる．本稿では上記の議論にもとづき，通常モードの区間において I) その継続時間と，II) その累積消費電力，すなわち消費エネルギーを特徴量として抽出する．

3.5 Step 5: 外れ値検知

Step 4 で抽出した特徴量にもとづき，外れ値検知アルゴリズムを適用して悪意のある機能が発現した通常モードの区間を検出する．本稿では，外れ値検知アルゴリズムとして局所外れ値因子 (LOF) 法 [7] を適用する．LOF では，周囲と比較して密度が極端に異なるデータを外れ値として識別する．それぞれのデータに対し LOF と呼ばれる値を算出するが，その値が 1 に近い場合は周囲のクラスタに属すると判定され，1 から大きく離れた場合には外れ値として判定される．

提案手法では，Step 4 で抽出された通常モードのそれぞれの区間における継続時間と消費エネルギーをもとに，外れ値を検知することで悪意のある機能の発現を検知する．通常モードと識別された区間における継続時間と消費エネルギーは，条件判定やループの数によってその時々で変化する．ただし，長期間動作させれば，同様の処理を行ったときの継続時間と消費エネルギーは，同様の値を示すはずである．このとき，密度に着目すればそれぞれのデータに対し周辺の密度はほぼ近い値を示す．ところが悪意のある機能が発現することで継続時間や消費エネルギーに明らかな変化が生じた場合，そのデータの密度は周辺のデータと比較して明らかに小さい値を示すはずである．この点において，LOF を用いた外れ値検知は本稿の提案手法に適している．

本稿の提案手法では，上記で述べたように，測定した消費電力波形から通常モードの区間を識別し，継続時間と消費エネルギーを特徴量として抽出する．抽出した特徴量に対し LOF 法を適用することで，悪意のある機能の発現を検知する．

4. 実験結果

本章では，3 章で提案した手法にもとづき，実際の装置

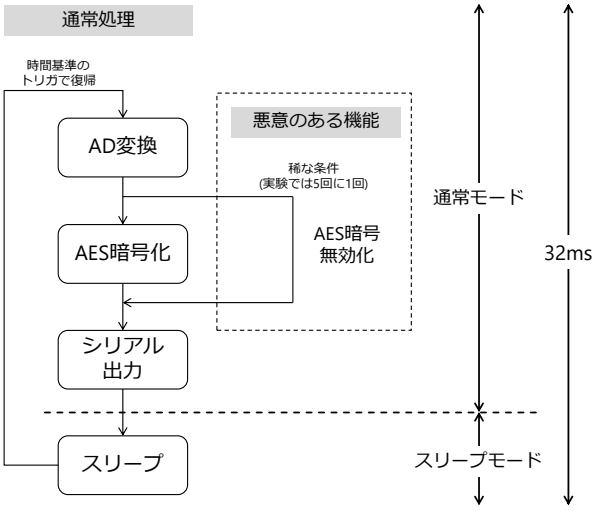

図 4　マイクロコントローラに実装したアプリケーションと悪意のある機能の概要.

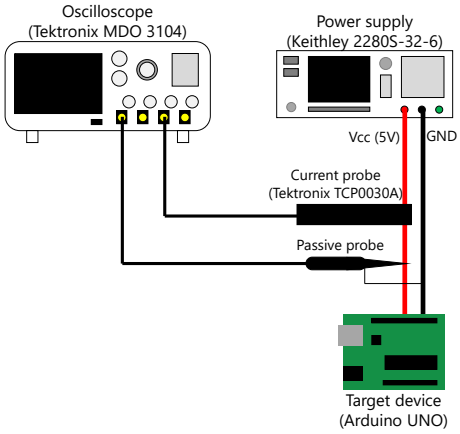

図 5　測定環境.

を用いて実験した結果を示す.

4.1　対象デバイスの実装

実験では，8 ビットマイクロコントローラ Microchip ATmega328P を搭載した，Arduino UNO ボードを利用する．アプリケーションは C++言語で実装した．アプリケーションでは，暗号化機能をもったセンサロガーを想定する．図 4 に，アプリケーションの概要を示す．通常モードにおける主な機能として，1) AD 変換，2) AES 暗号化，3) シリアル出力がある．AES 暗号処理には，C++言語で実装されたライブラリ [8] を利用する．これらの処理を終えると，マイクロコントローラはスリープモードに移行する．実装したプリケーションでは，32ms ごとに通常モードに復帰する．以上のアプリケーションに対し，悪意のある機能を挿入した．悪意のある機能では，5 回に 1 回 AES 暗号化処理を無効化し，AD 変換で得られた結果をそのままシリアル出力する.

4.2　実験環境

図 5 に実験環境を示す．電源装置として Keithley 2280S-32-6 を利用し，対象デバイスに電源を供給する．実験では電源電圧を 5V，最大電流を 0.4A と設定した．電力を測定するための装置として，オシロスコープ Tektronix MDO 3104 を利用した．電圧は受動プローブ，電流は電流プローブ TCP0030A を利用して測定した．電圧と電流を別々に測定し，オシロスコープの演算機能を利用して電圧と電流の積を取ることで消費電力を算出した.

オシロスコープで測定した消費電力にもとづき，Intel Core i7-5600U の CPU と 8GB のメモリを搭載したコンピュータ上で提案手法を適用した．提案手法は Python 3 で記述され，k 平均法と LOF にはデータ処理ライブラリ scikit-learn [9] を利用した.

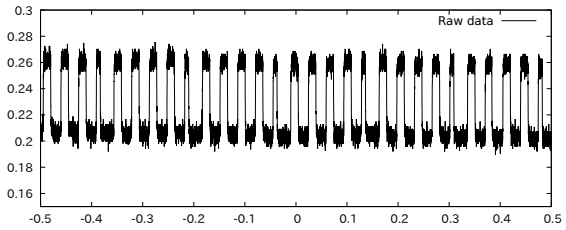

図 6　Step 1: 電力測定の結果.

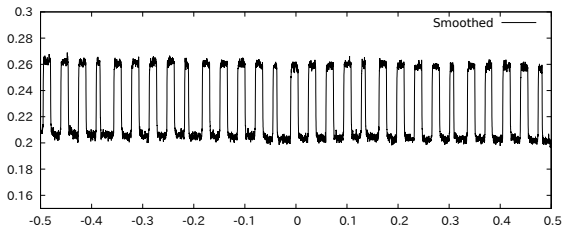

図 7　Step 2: 波形整形の結果.

4.3　実験結果

本節では，各ステップにおける実験結果を示す.

Step 1: 電力測定

図 5 に示した実験環境において，対象デバイスの消費電力を計測した．実験では，1 秒間の消費電力を計測した．図 6 に計測した消費電力を示す．図において，横軸は計測時間 [s]，縦軸は消費電力 [W] を示す．図 6 に示されるように，スリープモード時は 0.2 から 0.22W，通常モード時は 0.26 から 0.28W 程度の電力を消費する.

Step 2: 波形整形

Step 2 では，Step 1 で計測された波形を平滑化する．図 6 に示される波形を平滑化した結果を図 7 に示す．図において，横軸は計測時間 [s]，縦軸は消費電力 [W] を示す．図 7 に示されるように，図 6 と比べ波形に小刻みな波が少ない結果を得た.

Step 3: 通常/スリープモードの識別

Step 3 では，k 平均法を適用して通常モードとスリープモードを識別する．図 8 に識別結果を示す．図において，横軸は計測時間 [s]，縦軸は消費電力 [W] を示す．図 8 にお

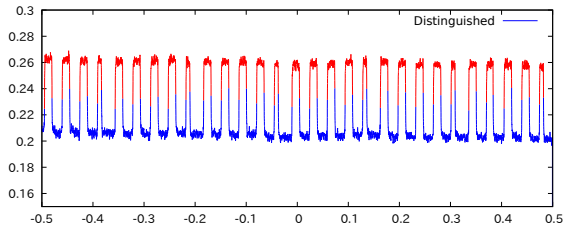

図 8 Step 3: 通常/スリープモードの識別の結果.

いて，赤色は通常モード，青色はスリープモードと識別された区間を示す．k 平均法を適用することで，通常モードとスリープモードを正しく識別できたことを確認できる．

Step 4: 特徴量抽出

ここではアクティブモードと識別された区間に着目する．それぞれの区間に対し，I) 継続時間と，II) 消費エネルギーを取得し，特徴量とする．表 1 に抽出された特徴量を示す．実験では，29 個の区間がアクティブモードと識別された．

Step 5: 外れ値検知

Step 4 で抽出された特徴量に対し LOF 法を適用することで，外れ値を検知する．LOF 法を適用した結果を，表 1 の「LOF」列に示す．表 1 において，区間 4, 9, 14, 19, 24, 29 の LOF はそれぞれ明らかに小さい値を示している．これらの結果は，LOF により外れ値として識別されたものである．ここで識別された外れ値を示す区間は実際に悪意のある機能が発現した区間であるため，実験により悪意のある機能の発現の検知に成功した．

5. まとめ

本稿では，マイクロコントローラのスリープモードに着目し，消費電力にもとづき悪意のある機能の発現を検知する手法を提案した．実験の結果，センサロガーを模した IoT デバイスに対し，悪意のある機能の発現検知に成功したことを確認した．

今後は，本稿の実験で使用したマイクロコントローラと異なるアーキテクチャのマイクロコントコーラや，異なるアプリケーションに挿入された悪意のある機能の発現検知を検証する．

謝辞

本研究開発は一部，総務省 SCOPE (受付番号 171503005) の委託を受けた．また第一著者は，JSPS 特別研究員奨励費 JP18J14325 の助成を受けた．

表 1 Step 4: 特徴量抽出と Step 5: 外れ値検知の結果.

区間	継続時間 [s]	消費エネルギー [mW · s]	LOF
1	0.0148	3.87	-1.42
2	0.0146	3.82	-0.94
3	0.0147	3.82	-1.01
4	0.0078	2.03	-19.25
5	0.0148	3.84	-1.14
6	0.0147	3.83	-1.02
7	0.0147	3.82	-0.95
8	0.0147	3.84	-1.10
9	0.0078	2.01	-19.33
10	0.0146	3.79	-0.93
11	0.0146	3.79	-0.94
12	0.0146	3.79	-0.93
13	0.0146	3.78	-0.98
14	0.0078	2.00	-19.40
15	0.0147	3.78	-1.00
16	0.0147	3.79	-0.94
17	0.0145	3.74	-1.12
18	0.0146	3.79	-0.96
19	0.0078	2.02	-19.29
20	0.0147	3.81	-0.94
21	0.0146	3.77	-1.00
22	0.0146	3.76	-1.01
23	0.0146	3.75	-1.02
24	0.0078	2.00	-19.40
25	0.0146	3.76	-1.00
26	0.013	3.35	-8.36
27	0.0146	3.76	-1.00
28	0.0146	3.75	-1.04
29	0.0078	1.99	-19.50

参考文献

[1] K. Zhao and L. Ge, “A survey on the Internet of things security,” in *Proc. International Conference on Computational Intelligence and Security*, 2013, pp. 663–667.

[2] J. Francq and F. Frick, “Introduction to hardware Trojan detection methods,” in *Proc. Design, Automation and Test in Europe (DATE)*, 2015, pp. 770–775.

[3] Y. Jin and Y. Makris, “Hardware Trojan detection using path delay fingerprint,” in *Proc. International Workshop on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[4] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, “A score-based classification method for identifying hardware-trojans at gate-level netlists,” in *Proc. Design, Automation and Test in Europe (DATE)*, 2015, pp. 465–470.

[5] R. Shende and D. D. Ambawade, “A side channel based power analysis technique for hardware trojan detection using statistical learning approach,” in *Proc. International Conference on Wireless and Optical Communications Networks (WOCN)*, 2016, pp. 1–4.

[6] J. Macqueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14, 1967, pp. 281–297.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: identifying density-based local outliers,” in *Proc. ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104.

[8] “AESLib.” https://github.com/DavyLandman/AESLib

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: machine learning in python,” *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.