損傷センサシステムにおける暗号のハードウェア実装

渡邉晃生*，　青木一弘**，早川潔**

Hardware Implementation of Encryption in Damage Sensing System

Kouki WATANABE*, Kazuhiro AOKI** and Kiyoshi HAYAKAWA**

要旨

日本の多くの構造物は老朽化が進んでおり，予算が限られる中，構造物を少しでも長く使用するためには，日常的な点検・維持管理が重要である．この点検・維持管理作業を効率化する手法として，構造物にセンサを設置して振動などの物理量を計測し，処理することにより構造物の損傷の位置や程度の同定を行う損傷センサシステムの研究開発が進められている．橋梁など大規模な構造物に対しこのシステムを導入する場合には，構造物の複数の個所にセンサを設置し，センサで計測したデータを通信手段により収集する必要がある．本研究では，損傷センサシステムにおける通信の暗号化について検討を行った．本稿では，暗号化に KCipher-2 を用いてハードウェアで実装を行うことで，処理の高速化が実現できることを示す．

キーワード: 暗号化, FPGA, KCipher-2, SHA256

1．はじめに

日本の社会基盤構造物の多くは，1970 年代および 1986 年から 90 年代頃に建設された．近年，従来では再開発・新設の対象となるような構造物が，経済の低迷や環境への配慮により補修・補強を施すことによる長寿命化が図られている．また，近年頻発している地震などの大規模災害や交通量の増大により，構造物の耐震性や耐荷力の向上も求められている．構造物の長寿命化のためには，日常的な点検・維持管理が重要である．現在における主流な点検手法は，熟練技術者による目視や打音による検査となっているが，需要の増加や専門技術者の減少，検査対象がインフラ構造物である場合は点検時に交通規制の必要があるなどといった問題がある．そこで，熟練した専門家を必要とせず，構造物の損傷を簡単に診断可能なシステムが求められている．以上の背景より，構造物にセンサを設置して振動などの物理量を計測し，信号処理などをすることにより構造物の損傷の位置や程度の同定を行う構造物ヘルスモニタリングシステムの研究開発が進められている．文献[1]において，圧電素子を振動センサとして用いて構造物に加わる振動を計測し，そのスペクトルの変化を観測することで損傷の位置や程度の同定がある程度可能であると報告されている．

本研究では，その提案を受けて開発された損傷センサシステム[2]における計測データのセキュリティについて検討を行う．先行研究[3]では，暗号化にはハッシュ関数によるストリーム暗号を用いたが，本研究では暗号化に KCipher-2[4]を用いてハードウェア実装を行うことで，処理をより高速化することを目的とした．

2．損傷センサシステム

2.1　システムの構成

本研究の損傷センサシステムは，複数の計測モジュール，収集モジュール，クラウドサーバから構成される．橋梁などの計測対象とする構造物に複数の計測モジュールと収集モジュールを設置し，振動センサとして圧電素子を用いて構造物に伝わる振動を計測し，計測モジュールによって波形観測及びスペクトル解析を行う．解析結果は自動的に収集モジュールに収集され，インターネットを通じてクラウドサーバにアップロードされる．このシステムを全国の構造物に配置することにより，クラウドサーバであらゆる構造物の損傷状態の一元管理を行う．システムの構成を図 1 に示す．

このシステムでデータの改ざんが行われると，構造物

2019 年 8 月 19 日 受理

* 総合工学システム専攻　電気電子工学コース（Advanced Course, Dept. of Technological Systems : Electrical and Electronic Engineering Course）

** 総合工学システム学科　電子情報コース（Dept. of Technological Systems : Electronics and Information Course）

の損傷を把握できないことや，不要な作業を行うことになるなど，様々な問題が発生することが考えられる．本研究では，計測モジュールと収集モジュール間において，計測したデータの盗聴や改ざん，送信者の成りすましについて対策を行うために，クラウドサーバへアップロードする前に計測モジュール上でデータの暗号化とメッセージ認証コードの作成を行う．

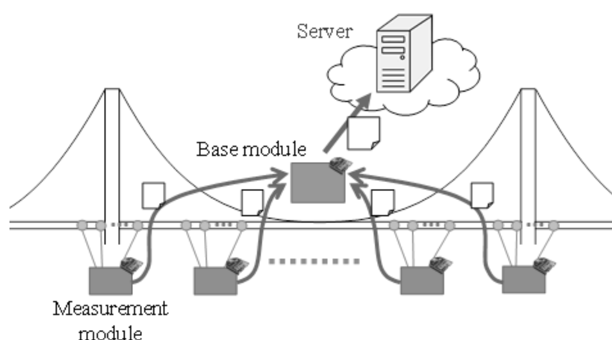

図１　損傷センサシステムの概要

2.2　計測モジュール

計測モジュールには，Xilinx 社の FPGA 開発ボード ZYBO を用いた．ZYBO の外観を図２に示す．

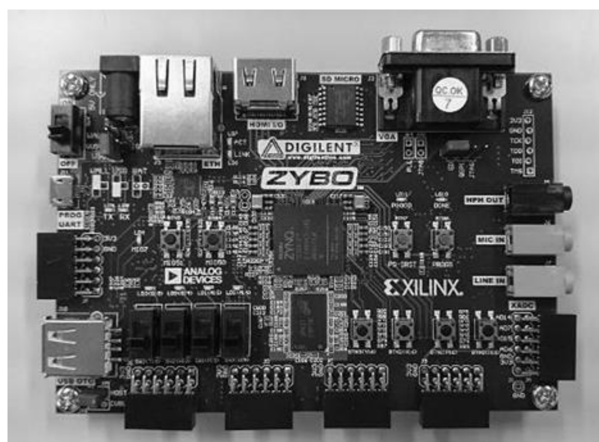

図２　ZYBO の外観

ZYBO は Zynq 7010 を搭載したデバイスで，ARM プロセッサ(Cortex-A9)と FPGA(Artix-7 相当)が内蔵されている．本研究では ARM 上で動作する OS として Linaro Ubuntu 12.11 をインストールした ZYBO に，データの暗号化とメッセージ認証コード作成の処理を実装した．また，ZYBO の FPGA 部は，6 入力 LUT4 つとフリップフロップ 8 つで構成されるスライスが 4400 個，32[kbyte]のデュアルポートブロック RAM80 個などから構成されている．

計測モジュールと収集モジュール間の通信には，XBee ZB を用いて ZigBee による無線通信を行う．

2.3　暗号化方式

計測モジュールで用いる暗号は，低消費電力で高速に演算できる必要がある．そこで本研究では，処理手順を抑えて高速化が容易な共通鍵暗号方式を用いた．共通鍵暗号方式には，ブロック暗号とストリーム暗号の二つの方式がある．ストリーム暗号は，生成された擬似乱数列と平文を XOR 演算する処理だけで暗号化するため，演算量を抑えることができる．

これらの理由から，本研究では，処理速度，省電力性に優れたストリーム暗号を用いた．暗号化アルゴリズムには KCipher-2 を用いた．

３．KCipher-2

KCipher-2[4,5]は，128[bit]の初期鍵と 128[bit]の初期ベクトルを入力とするストリーム暗号である．暗号技術検討会及び関連委員会(CRYPTREC)の電子政府推奨暗号に選定されたストリーム暗号であり[6]，2 つのフィードバックレジスタ *FSR-A* と *FSR-B*，4 つの内部レジスタを有する非線形関数部，動的フィードバック制御部から構成されている．内部で使用されるレジスタは全て 32[bit]である．KCipher-2 のアルゴリズムの概要を図３に示す．

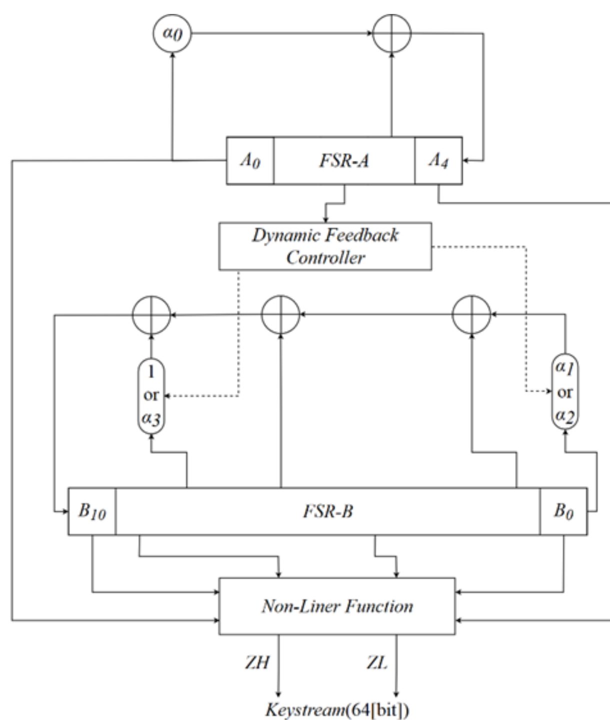

図３　KCipher-2

3.1　初期化処理

KCipher-2 の初期化処理は，初期鍵・初期ベクトルの読込み処理と内部状態の初期化ステップからなる．

3.1.1　初期鍵・初期ベクトルの読込み

初期鍵・初期ベクトルの読込みでは，鍵スケジュールアルゴリズムを用いて，128[bit]の初期鍵 *IK* と 128[bit]の初期ベクトル *IV* を内部レジスタに読込み，初期化前の内部状態を生成する．

KCipher-2 の鍵スケジュールアルゴリズムでは，128[bit]の初期鍵を 384[bit]の内部鍵に拡張する．IK_i，K_i を 32[bit]とし，初期鍵を $IK=(IK_0,IK_1,IK_2,IK_3)$，内部鍵を $K=(K_0,K_1,\cdots,K_{11})$と表すと，*K* は *IK* より以下の式で表される．

$$K_i = IK_i \quad (0 \leqq i \leqq 3) \quad (1)$$

$$K_i = K_{i-4} \oplus Sub\big((K_{i-1} \ll 8) \oplus (K_{i-1} \gg 24)\big) \oplus Rcon[\tfrac{i}{4}-1]) \quad (i=4,8) \quad (2)$$

$$K_i = K_{i-4} \oplus K_{i-1} \quad (\text{otherwise}) \quad (3)$$

ここで，$i=0,1,\cdots,11$ である．また，$Rcon[j]$は KCipher-2 の鍵スケジュールアルゴリズムでは$i=4,8$のみ利用される定数であり，$i=4$ の時，$Rcon[0]$=0x01000000，$i=8$ の時，$Rcon[1]$=0x02000000 となる．

内部鍵 *K* の生成後，初期化前の *FSR-A*，*FSR-B* の内部状態 *A*，*B* と非線形関数内部レジスタの値 *R*1，*R*2，*L*1，*L*2 は, 内部鍵 *K* と初期ベクトル$IV = (IV_0, IV_1, IV_2, IV_3)$（$IV_i$ は 32[bit]）から以下の式で生成される．

$$A_m = K_{4-m} \quad (m=0,1,\cdots,4) \quad (4)$$

$$B = (K_{10}, K_{11}, IV_0, IV_1, K_8, K_9, IV_2, IV_3, K_7, K_5, K_6) \quad (5)$$

$$R1 = R2 = L1 = L2 = 0\text{x}00 \quad (6)$$

3.1.2　内部状態の初期化

初期化前の内部状態を生成した後，KCipher-2 を 24 クロック($j=1,2,\cdots,24$)動作させ，以下の式により内部状態の撹拌を行う．

$$R1^{(j)} = Sub(L2^{(j-1)} + B_9^{(j-1)}) \quad (7)$$

$$R2^{(j)} = Sub(R1^{(j-1)}) \quad (8)$$

$$L1^{(j)} = Sub(R2^{(j-1)} + B_4^{(j-1)}) \quad (9)$$

$$L2^{(j)} = Sub(L1^{(j-1)}) \quad (10)$$

$$A_i^{(j)} = A_{i+1}^{(j-1)} \quad (i \neq 4) \quad (11)$$

$$A_i^{(j)} = \alpha_0 A_0^{(j-1)} \oplus A_3^{(j-1)} \oplus ZL^{(j-1)} \quad (i=4) \quad (12)$$

$$B_i^{(j)} = B_{i+1}^{(j-1)} \quad (i \neq 10) \quad (13)$$

$$B_i^{(j)} = \left(\alpha_1^{cl1^{(j-1)}} + \alpha_2^{1-cl1^{(j-1)}} - 1\right) B_0^{(j-1)} \oplus B_1^{(j-1)} \oplus B_6^{(j-1)} \oplus \alpha_3^{cl2^{(j-1)}} B_8^{(j-1)} \oplus ZH^{(j-1)} \quad (i=10) \quad (14)$$

ここで, 時刻 *x* の *FSR-A*, *FSR-B* の出力を $A^{(x)}$, $B^{(x)}$とし，非線形関数内部レジスタの値も同様に $R1^{(x)}$，$R2^{(x)}$，$L1^{(x)}$，$L2^{(x)}$としている. また, *i* はレジスタの番号を示している．$Sub(X)$は入力 *X* に対する関数 *Sub* の出力を表し，関数 *Sub* は 32[bit]から 32[bit]への変換を与える．

*cl*1，*cl*2 は *FSR-A* のレジスタ値を用いてフィードバック制御部により決定される 1[bit]の値である．$A_i^{(x)}[y]$を時刻 *x* の A_i の *y* 番目のビット値とし，MSB を $A_i[31]$とすると，*cl*1，*cl*2 は以下の式で表される．

$$cl1^{(x)} = A_2^{(x)}[30] \quad (15)$$

$$cl2^{(x)} = A_2^{(x)}[31] \quad (16)$$

3.2　鍵系列出力処理

時刻 *t* に出力される 64[bit] の鍵系列を$Z^{(t)} = (ZH^{(t)}, ZL^{(t)})$と表す. $ZH^{(t)}$, $ZL^{(t)}$は 32[bit]で, $ZH^{(t)}$が上位の 32[bit]である．$ZH^{(t)}$，$ZL^{(t)}$は以下の式で出力される．

$$ZH^{(t)} = B_{10}^{(t)} + L2^{(t)} \oplus L1^{(t)} \oplus A_0^{(t)} \quad (17)$$

$$ZL^{(t)} = B_0^{(t)} + R2^{(t)} \oplus R1^{(t)} \oplus A_4^{(t)} \quad (18)$$

鍵系列出力後，以下の式により内部状態の更新を行い，時刻 t+1 の内部レジスタ値を得る．なお，*R*1，*R*2，*L*1，*L*2 の更新については式(7)～(10)と同様に行う．

$$A_i^{(t+1)} = A_{i+1}^{(t)} \quad (i \neq 4) \quad (19)$$

$$A_i^{(t+1)} = \alpha_0 A_0^{(t)} \oplus A_3^{(t)} \quad (i=4) \quad (20)$$

$$B_i^{(t+1)} = B_{i+1}^{(t)} \quad (i \neq 10) \quad (21)$$

$$B_i^{(t+1)} = \left(\alpha_1^{cl1^{(t)}} + \alpha_2^{1-cl1^{(t)}} - 1\right) B_0^{(t)} \oplus B_1^{(t)} \oplus B_6^{(t)} \oplus \alpha_3^{cl2^{(t)}} B_8^{(t)} \quad (i=10) \quad (22)$$

3.3　安全性評価

KCipher-2 に対する 2^{256} 以下の計算量で実行できる具体

的な攻撃手法は提案されていない．表１にいくつかのストリーム暗号について効果的な攻撃に要する計算量を示す[7]．表１より，ほかの暗号と比較しても，KCipher-2 への攻撃に要する計算量は多くなっており，安全であることがわかる．

表１　安全性の比較

Algorithm	Complexity of the best attack	Kind of attack
KCipher-2	2^{320}	guess-and-determine
Rabbit	2^{136}	distinguishing
SNOW 2.0	2^{174}	distinguishing
SOSEMANUK	2^{148}	linear
HC-256	$2^{276.8}$	distinguishing
Salsa20	2^{224}	related-cipher
SNOW 3G	2^{320}	guess-and-determine

４．メッセージ認証

データの暗号化は盗聴防止に対しては有効であるが，改ざんの検出や送信者のなりすましについては他の対策が必要となる．そこで本研究ではメッセージ認証を用いることで対策を行った．

メッセージ認証では，送信者は，暗号文と共通鍵を結合したビット列からハッシュ関数を用いてメッセージ認証コード(MAC)を作成し，暗号文に付加して送信する．受信者は，送られてきた暗号文から MAC を送信者と同様の手順で作成し，受信した MAC と作成した MAC が等しければ，送信者は正しく，データの改ざんが行われていないと確認できる．本研究ではハッシュ関数に SHA256[8]を用いた．

５．SHA256

SHA256 は，任意長の入力データから 256[bit]のハッシュ値を出力するハッシュ関数である．256[bit]の初期ハッシュ値を持っており，入力データを 64[byte]毎のメッセージブロック$M^{(i)}$に分割し，初期ハッシュ値を変化させる演算に各メッセージブロックを適用させることで入力データ長に関係なく固定長のハッシュ値を得ることができる．

入力データに対しては，64[byte]の倍数のデータを得るようにパディング処理を行った．処理の概要を図４に示す．この処理では，最初に元の入力データの後に 1 のビットと元の入力データ長を 2 進数表記した 8[byte]のデータを追加する．そして，データ全体のサイズが 64[byte]の倍数となるまで，追加した 1 のビットと入力データ長の間に 0 のビットを追加することによって，64[byte]の倍数のデータを得ることができる．

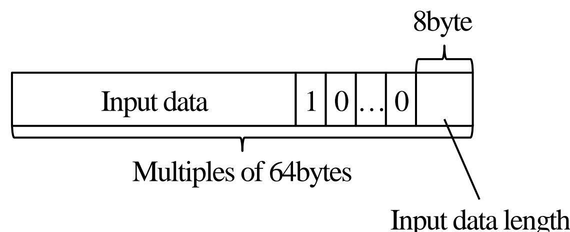

図４　SHA256 のパディング処理

入力データを 64[byte]毎に分割したメッセージブロック $M^{(i)}$をさらに 32[bit]毎に分割して$\left(M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}\right)$とし，このデータ$M_t^{(i)}$を 256[byte]のデータに拡張してハッシュ値の計算に用いる．拡張されたデータは 32[bit]毎に$(W_0, W_1, \ldots, W_{63})$と表記する．拡張されたデータは以下の式で表される．

$$W_t = M_t^{(i)} \quad (0 \leqq t \leqq 15) \qquad (23)$$

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \quad (16 \leqq t \leqq 63) \qquad (24)$$

ここで，$ROTR^n(x)$を x の右 n ビットローテート，$SHR^n(x)$を x の右 n ビットシフトとすると$\sigma_0(x), \sigma_1(x)$は以下の式で表される．

$$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \qquad (25)$$

$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \qquad (26)$$

６．暗号化と MAC の処理

6.1　ソフトウェア実装

本研究では，MicroSD カード上に保存された 6400[byte]の計測データファイルを暗号化し，32[byte]の MAC を作成し付加したファイルを出力するプログラムを C 言語で作成した．図５に，暗号化処理後のファイル構成を示す．

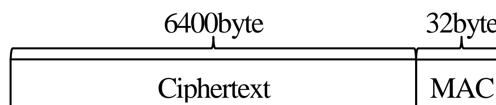

図５　暗号化処理後のファイル構成

鍵は 32[byte]のファイルを用い，プログラム中で生成される擬似乱数列によって一つのファイルを暗号化する毎に更新している．そのため，内容が同じファイ

ルを続けて入力しても，異なる暗号文が出力される．暗号化処理の流れを図６に示す.

今回行った実装では，KCipher-2 の実装に参照テーブルを用いた高速実装方法[4]を用いている. この方法ではα_iの乗算や関数 *Sub* の処理を，参照テーブルを用いて行うため高速となる．また，入力される平文と鍵が固定長であるため, SHA256 のパディング処理を定数によって行うことで簡略化し，プログラムの高速化を行っている.

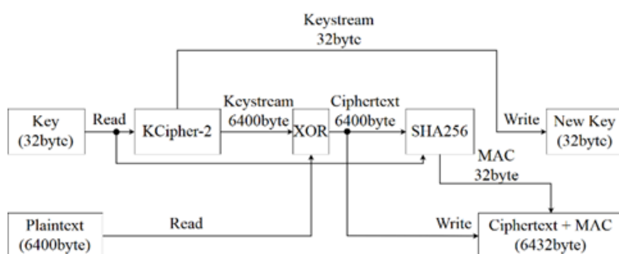

図６　暗号化処理の構成

6.2　ハードウェア実装

ハードウェア実装では, 図６のデータ暗号化部と MAC 作成部を FPGA 上に実装した. 開発には Xilinx 社の FPGA 開発ツール Vivado を用いて Verilog-HDL により記述している．ARM 上から FPGA 上のレジスタアクセスには, UIO(User space I/O)を用いている．この方法では，デバイスツリーに少しの記述を行うだけでユーザー空間からのレジスタアクセスが可能となる．今回は UIO を２つ用いて 32[bit]単位でデータの入出力を行っている．図７において，FPGA で囲んでいる部分がハードウェアで実装した部分を示している.

また, ハードウェア実装では, SHA256 の処理でハッシュ値の計算に用いる 256[byte]のデータ($W_0, W_1, \dots, W_{63}$)の保存場所を，レジスタからブロック RAM へと置換することによって，回路規模の削減を図った.

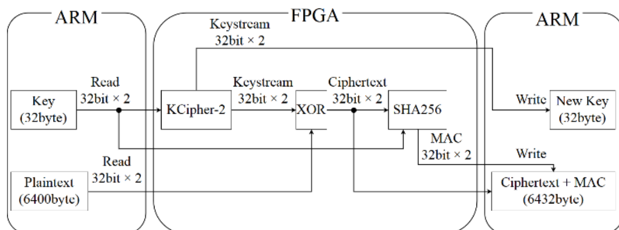

図７　ハードウェア実装による暗号化処理の構成

7．性能の比較

7.1　擬似乱数列生成速度の比較

本研究で用いるストリーム暗号の性能比較を行うため, 先行研究で用いたハッシュ関数によるストリーム暗号のソフトウェア実装[3]，KCipher-2 のソフトウェア実装，および KCipher-2 のハードウェア実装における, 暗号化と鍵の更新に必要な 6432[byte]の擬似乱数列の処理時間を比較した．結果を表２に示す．KCipher-2 のハードウェア実装においては，クロック周波数を 125[MHz]とし，擬似乱数列生成に必要なクロック数から処理時間を算出した．表２より，ハッシュ関数を用いる既存手法よりも KCipher-2 を用いた方が高速に擬似乱数列の生成ができることがわかった．また，FPGA でハードウェア実装することによりさらに高速化できることがわかった.

表２　処理時間の比較

	Processing time [μsec]
Hash Function (Software Imp.)	12707
KCipher-2 (Software Imp.)	740
KCipher-2 (Hardware Imp.)	53

7.2　暗号化プログラムの実装別比較

KCipher-2 による暗号化と MAC の作成について，ソフトウェア実装(ソフト実装)，ハードウェア実装(ハード実装 1)，SHA256 の処理部分のレジスタを一部ブロック RAM に置換した実装(ハード実装 2)についての 3 種類の実装を行い，それぞれの実行時間，FPGA の回路規模, ZYBO 全体の消費電力について比較を行った．結果を表３に示す.

表３　実装方法による性能の比較

	Software Imp.	Hardware Imp.1	Hardware Imp.2
Processing time [μsec]	5266	3095	3191
Slice	0 (0%)	2456 (55.8%)	1333 (30.3%)
Block RAM	0 (0%)	10.5 (17.5%)	11.5 (19.2%)
Ordinary power [W]	1.7747	1.8185	1.8034
Encryption power [W]	1.8130	1.8669	1.8505

回路規模の比較は, 使用したスライスとブロック RAM で行い，使用率も併記している．ブロック RAM が小数となっているのは, ブロック RAM は１つにつき 32[kbyte]のデュアルポートであるが，一部のブロック RAM を 16[kbyte]のシングルポート ROM として用いているためである.

消費電力は，暗号化をしない平常時の消費電力と暗号化時の消費電力の平均で比較している．消費電力の測定には，日置電機のパワーメータ PW3335-03 を用いた．サンプリング間隔は 200[msec]で 20 分間の測定を行い，1 秒毎に暗号化処理を行うプログラムを実行した．図８にソフトウェア実装時における電力波形の一例を示す．

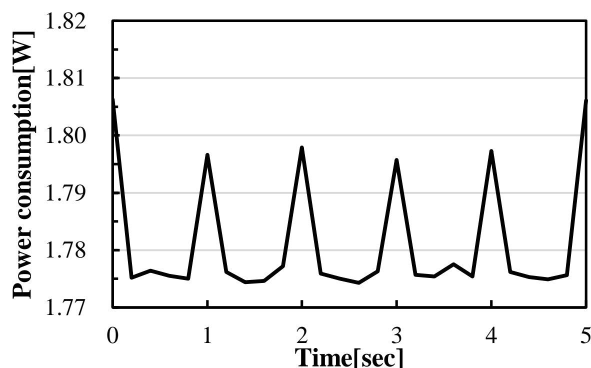

図８　ソフトウェア実装時における消費電力

表３より，FPGA でハードウェア実装することで処理が高速化できていることがわかる．ハード実装 1 に比べハード実装 2 が少し遅いのはブロック RAM を用いたことでデータへのアクセスに必要なクロック数が増えたためである．しかし，回路規模で比較した場合，スライス数がかなり削減されており効率的な回路になっていると考えられる．

消費電力で見れば，ソフト実装の場合が最も低く，ハード実装 2, ハード実装 1 の順に高くなっていることがわかる．これはいずれの実装においても ARM プロセッサは動作しており，ハードウェア実装ではさらに FPGA により回路が実装されている分の消費電力が加えられているためだと考えられる．ハード実装 1 に比べてハード実装 2 が少し低いのは，回路規模が小さくなっているためだと考えられる．そのためハードウェア実装において消費電力を低減するには，回路規模の削減が重要であると考えられる．また，平常時と比べ暗号化時に増加する消費電力はいずれの実装の場合もほぼ同じであることがわかる．

８．むすび

本研究では，損傷センサシステムに暗号化処理を実装した．暗号化方式には共通鍵暗号方式のストリーム暗号を選択し，KCipher-2 によるデータの暗号化と，SHA256 によるメッセージ認証を行った．実装方法としては，ソフトウェア実装，ハードウェア実装，SHA256 の処理部分のレジスタを一部ブロック RAM に置換したハードウェア実装の 3 種類について行った．実験の結果より，ハードウェア実装を行うことで高速な処理を実現することができた．消費電力においては，ハードウェア実装を行うことで若干増加したが，回路規模の削減を行うことで低減できることを確認した．

今後は，ハードウェア実装における回路規模のさらなる削減を行うことで，消費電力を増やさずに暗号化処理の高速化を実現していく必要がある．

参考文献

[1] 小幡卓司，2014，圧電素子を用いた損傷同定モニタリングシステムの実験的研究，構造工学論文集，Vol60A，pp.165-174.

[2] Kiyoshi HAYAKAWA，Masashi FUJIWARA，Takeshi WADA，Takashi OBATA，2016，Development of a damage identification monitoring system for building structures，Proceedings of the 4th IIAE International Conference on Industrial Application Engineering 2016，pp.340-347.

[3] 渡邊晃生，青木一弘，早川潔，三野智貴，中野郁弥，2016，損傷センサシステムにおけるセキュリティの検討，第 35 回数理科学講演会講演論文集，B101.

[4] KDDI 株式会社，2017，ストリーム暗号 KCipher-2(仕様書 1.2 版).

[5] 響崇史，本間尚文，青木孝文，仲野有登，福島和英，清本晋作，三宅優，2012，KCipher-2 への電力解析攻撃対策とその評価，Computer Security Symposium，pp.749-756.

[6] 総務省，経済産業省，2016，電子政府における調達のために参照すべき暗号のリスト(CRYPTREC 暗号リスト)，http://www.cryptrec.go.jp/list/cryptrec-ls-0001-2016.pdf.

[7] Yuto NAKANO，Kazuhide FUKUSHIMA，Shinsaku KIYOMOTO，Tsukasa ISHIGURO，Yutaka MIYAKE，Toshiaki TANAKA，Kouichi SAKURAI，2014，Fast Implementation of KCipher-2 for Software and Hardware，IEICE TRANS. INF. & SYST., VOL.E97-D, NO.1，pp.43-52.

[8] NIST，2015，Secure Hash Standard (SHS)，http://dx.doi.org/10.6028/NIST.FIPS.180-4.