# Authenticatie voor the internet of things

Tim Cooijmans T.J.P.M.Cooijmans@student.ru.nl
Manu Drijvers ManuDrijvers@student.ru.nl
Patrick Verleg P.Verleg@student.ru.nl

30 juni 2010

## 1 Inleiding

### 1.1 Onderwerp

In kader van het vak research and development hebben wij the internet of things bestudeerd. We zien erg veel potentie in dit concept en het concept is zelfs erg goed te realiseren. Voordat het concept werkelijkheid kan worden, zijn er een aantal problemen die opgelost moeten worden. Deze problemen zijn voornamelijk op het gebied van privacy en beveiliging. Voordat we deze problemen gaan behandelen zullen we eerst het basis idee van the internet of things uitleggen.

### 1.2 The internet of things

Het idee achter the internet of things is dat steeds meer simpele dingen in staat worden gesteld om met elkaar te communiceren. Ze kunnnen informatie delen met elkaar of een wereldwijde schaal. Dit wil zeggen dat bijvoorbeeld een ding in Nederland direct informatie kan delen met een ding in de Verenigde Staten. Een voorbeeld van een simpele toepassing van deze techniek is als volgt:

Stel je voor, je ligt te slapen. Plotseling gaat je wekker af, zonder dat jij die de avond te voren hebt gezet. De wekker heeft namelijk in je agenda gekeken, en gezien dat je eerste afspraak om 10.45 in het Huygensgebouw is. Hij heeft bepaald dat de reistijd vanaf je huis 20 minuten is, en weet dat je een uur nodig heb om op te staan. Dus om 9.25 begint je wekker te rinkelen, en gaan meteen de lampen branden en staat de verwarming al hoog. Als je na een uurtje de deur uit gaat, merkt de deur dat je vertrekt, en laat het de lampen en verwarming weten dat ze weer uit mogen.

### 1.3 Problemen

Zoals eerder besproken zijn er een aantal problemen: Allereerst is het zonder encryptie mogelijk om dataverkeer tussen artefacten af te luisteren. Dit is natuurlijk natuurlijk niet de bedoeling gezien deze data privégegevens kan bevatten. Een ander probleem is dat het zonder een goede methode om de authenticiteit van een afzender te controleren, mogelijk is dat artefacten andere artefacten ongeauthenticeerd aansturen. Je wil bijvoorbeeld niet dat een ander zomaar je TV uitschakelt.

Een oplossing voor dit probleem is het signen van berichten met behulp van een hash. Hiervoor worden hash-functies gebruikt. De verzender en ontvanger passen beide deze functie toe op een string bestaande uit de sleutel van de ontvanger en het tijdstip waarop het bericht is verzonden. Een goede hashfunctie is niet te kraken in redelijke tijd met de huidige rekenkracht van computers. Een andere eis aan een hashfunctie is dat het niet veel rekentijd kost om de functie toe te passen. Artefacten in the internet of things hebben over het algemeen weinig rekenkracht en je wilt niet dat het lang duurt om een pakket te verzenden of ontvangen. Ook energieverbruik is een belangrijk punt: Als elk artefact deel is van the internet of things is het niet wenselijk dat elk artefact veel energie verbruikt. Wij gaan onderzoeken welk van de gangbare hash-functies veilig is en van de veilige hashing algoritmes het minste processortijd kost, waarbij veilig als volgt gedefineerd is: Het achterhalen van de sleutel op een computer met een x86-processor met een frequentie van 4 GHz moet gemiddeld minimaal 1 jaar duren.

# 2 Theoretisch kader

## 2.1 Kandidaat hash-functies

Wij hebben ons gelimiteerd tot de volgende hash-functies: MD5 en SHA-1 beide toegepast op de volgende string:
`[128 bits key][32 bits tijd]`

## 2.2 Achterhalen van de sleutel

Omdat MD5[4] en SHA1[1] beide niet injectief zijn, zijn er verschillende strings die naar dezelfde hash gemapt worden[5]. Dit zijn collisions. Normaal gesproken zijn collisions in de beveiliging een probleem. In ons geval zou het kunnen voor komen dat een ongeauthoriseerde zender zijn pakket signeert met een verkeerde sleutel, maar dat deze sleutel toch goedgekeurd wordt omdat de hash van deze verkeerde sleutel hetzelfde is als de hash van de echte sleutel. Toch is dit voor ons geen groot probleem, omdat de hash afhankelijk is van de tijd: Als het pakket met een andere timestamp gesigned wordt geven de juiste sleutel en de foute sleutel compleet andere hashes door het avalanche-effect in SHA1[1] en MD5[4]

Om de sleutel te achterhalen zonder voorkennis, moet je dus simpelweg dingen proberen om de echte sleutel te vinden. Dit is een zogenaamde brute-force attack. Omdat er $2^{128}$ mogelijke sleutels zijn bij zowel MD5 als SHA1, heb je na $2^{128}$ pogingen de zeker de echte sleutel achterhaald, er vanuitgaande dat je elke keer een andere sleutel probeert. De kans dat je de echte sleutel pas na $2^{128}$ pogingen raadt, is natuurlijk erg klein. Als je de helft van de sleutels geprobeerd hebt, is de kans $\frac{1}{2}$ dat je de echte sleutel hebt achterhaald.

Gemiddeld kost het je dus $\frac{1}{2} \cdot 2^{128} = 2^{127}$ pogingen om de sleutel te achterhalen, voor zowel MD5 als SHA1. Een x86 processor zal altijd minstens één instructie nodig hebben om een MD5 of SHA1 hash te berekenen. Op een 4 GHz computer maakt $4 \cdot 10^9 \cdot 60 \cdot 60 \cdot 24 \cdot 365 = 126144 \cdot 10^{12}$ berekeningen per jaar. Dit is $\frac{2^{127}}{126144 \cdot 10^{12}} \approx 10^{21}$ jaar. Beide hash-functies voldoen dus ruimschoots

aan onze veiligheidseis (zie 1.3).

### 2.3 Energieverbruik

Eerder hebben wij genoemd dat een laag energieverbruik belangrijk is voor the internet of things, omdat je zo veel artefacten hebt en je niet wil dat die allemaal constant veel verbruiken. Het energieverbuik hangt af van wat de microcontroller aan het doen is. De microcontroller heeft twee toestanden: running en sleep. Als er iets te berekenen valt is het in runningstate, anders in sleep. Het energieverbruik in runningstate is gemiddeld 6,4 $m$A, in sleep is dat minder dan 0,2 $\mu$A [3]. Het verbruik in runningstate verschilt per instructie, maar dat verschil is relatief erg klein [2]. Daarom hangt het energieverbruik bijna helemaal af van de processortijd van een programma.

Hierdoor is ons onderzoek naar processortijd voldoende om ook een uitspraak te kunnen doen over het energieverbruik.

## 3 Methode

Om te kunnen onderzoeken welk van de twee eerdergenoemde authenticatiemethoden het minste processortijd verbruikt, gebruiken wij een MicroChip PIC18F452. Dit is een 20MHz 8-bit microcontroller met 1536 bytes RAM en 32K flash memory[3]. Eerst moesten we MD5 en SHA1 uitvoerbaar maken voor de microcontroller. Hiervoor hebben we uit de TCP/IP libraries van Microchip de MD5 en SHA1 functies gehaald, en ze omgeprogrammeerd zodat ze los uitgevoerd kunnen worden. Omdat het werkgeheugen erg beperkt is en MD5 en SHA1 niet ontworpen zijn voor microcontrollers, was het nodig om de code te optimaliseren. De originele code zorgde voor stack overflow door te veel geneste functie aanroepen. Dit hebben we opgelost door code te kopiëren in plaats van functies aan te roepen. We hebben voor beide hashingfuncties een testprogramma geschreven dat het volgende doet:

- Het programma wordt geïnitialiseerd, de LED is uit.
- Het programma staat klaar om hashing iteraties uit te voeren, de LED gaat aan.
- Het programma gaat een bepaalde string een bepaald aantal keer hashen met MD5 of SHA1.
- Het programma is klaar met hashen, de LED gaat uit.

Hiermee kunnen we meten hoe lang het duurt om een bepaald aantal keer een bepaalde string te hashen met MD5 of SHA1.

We zijn als volgt te werk gegaan: Wij filmen de microcontroller met de LED, dan voeren we een bepaalde opdracht uit, bijvoorbeeld tweehonderd keer MD5. Als dit klaar is kijken we op de video op welke frame de LED aan gaat, en op welke frame de LED uitgaat. Zo kunnen we redelijk precies de tijd bepalen die nodig was om deze hashing opdracht uit te voeren.

We hebben beide hash functies met 100 en 200 iteraties tien keer uitgevoerd. Aan de hand hiervan kunnen we bepalen wat een hash-iteratie voor beide algoritmen kost zonder dat de resultaten beïnvloed worden door eventuele berekenin-

gen buiten de iteraties. Dit kan door de gemiddelde processortijd van 200 iteraties te verminderen met die van 100 iteraties en vervolgens te delen door 100.

# 4 Resultaten

Testresultaten MD5:

| testnummer | MD5 $i = 100$ (ms) | MD5 $i = 200$ (ms) |
|---|---|---|
| 1 | 0,754 | 1,467 |
| 2 | 0,776 | 1,488 |
| 3 | 0,743 | 1,470 |
| 4 | 0,770 | 1,419 |
| 5 | 0,792 | 1,492 |
| 6 | 0,742 | 1,477 |
| 7 | 0,765 | 1,465 |
| 8 | 0,753 | 1,470 |
| 9 | 0,733 | 1,453 |
| 10 | 0,795 | 1,433 |

Testresultaten SHA1:

| testnummer | SHA1 $i = 100$ (ms) | SHA1 $i = 200$ (ms) |
|---|---|---|
| 1 | 1,713 | 3,375 |
| 2 | 1,745 | 3,353 |
| 3 | 1,767 | 3,396 |
| 4 | 1,728 | 3,431 |
| 5 | 1,752 | 3,468 |
| 6 | 1,732 | 3,410 |
| 7 | 1,780 | 3,433 |
| 8 | 1,721 | 3,392 |
| 9 | 1,753 | 3,404 |
| 10 | 1,774 | 3,386 |

Statistische functies MD5:

| | MD5 $i = 100$ (ms) | MD5 $i = 200$ (ms) |
|---|---|---|
| gem. | 0,762 | 1,463 |
| gem. abs. afw. | 0,017 | 0,017 |
| std. afw. | 0,021 | 0,023 |

Statistische functies SHA1:

| | SHA1 $i = 100$ (ms) | SHA1 $i = 200$ (ms) |
|---|---|---|
| gem. | 1,747 | 3,405 |
| gem. abs. afw. | 0,019 | 0,024 |
| std. afw. | 0,023 | 0,033 |

MD5 1 iteratie: $\frac{1,463-0,762}{100} = 7,01 \cdot 10^{-3} ms$
SHA1 1 iteratie: $\frac{3,405-1,747}{100} = 16,58 \cdot 10^{-3} ms$

# 5 Discussie

Wij gaan er vanuit dat de verbinding veilig is. Met andere woorden, pakketten kunnen niet door anderen gelezen worden. Als dit niet het geval is, zou je het

verkeer kunnen afluisteren, en zolang een hash voor een bepaald artefact geldig is, pakketen met andere inhoud kunnen sturen naar dat artefact, met deze afgeluisterde hash als handtekening. In praktijk kun je niet altijd garanderen dat een verbinding veilig is.

Het onderzoek is slechts op één microprocessor uitgevoerd. Een andere microprocessor met een andere instructieset zou andere resultaten kunnen opleveren. Echter gebruikt het overgrote deel van de embedded devices een microcontroller met een vergelijkbare instructieset.

# 6 Conclusie

Uit onze resultaten blijkt dat een MD5 hash berekenen $7,01 \cdot 10^{-3} ms$ kost en dat bij een SHA1 hash $16,58 \cdot 10^{-3} ms$ is. Beide algoritmen voldoen aan onze gestelde veiligheidseis. Het energieverbruik zal bij MD5 lager zijn omdat de processortijd kleiner is (zie 2.3).

Gezien deze resultaten blijkt dat MD5 de meeste geschikte hashfunctie is voor authenticatie binnen the internet of things.

# Referenties


[1] D Eastlake and P Jones. RFC3174: US Secure Hash Algorithm 1 (SHA1). Technical report, RFC Editor United States, 2001.

[2] Mark Hempstead, Michael J. Lyons, David Brooks, and Gu-Yeon Wei. Survey of hardware systems for wireless sensor networks. *Journal of Low Power Electronics*, Volume 4, 2008.

[3] Microchip Technology Inc. *PIC18FXX2 Datasheet*, 2006.

[4] R Rivest. RFC1321: The MD5 message-digest algorithm. Technical report, RFC Editor United States, 1992.

[5] Wojciech A. Trybulec. Pigeon hole principle. *Journal of Formalized Mathematics*, Volume 2, 1990.

# A Bijlage 1: `main.c`

```
#define STACK_USE_MD5

#include <p18f4520.h>
#include <string.h>
#include "Hashes.h"

#pragma config OSC = HS
#pragma config PWRT = OFF
#pragma config BOREN = OFF
#pragma config WDT = OFF
#pragma config MCLRE = ON
#pragma config PBADEN = OFF
#pragma config LVP = OFF

HASH_SUM hashsum;
unsigned char string[4] = {'t','e','s','t'};
unsigned char hash[33];

void init(void)
{
    CMCON=0b00000111; // Close Comparator

    //I/O:  76543210 1=input, 0=output
    TRISA=0b00000000; // Configure PORTA I/O
    TRISB=0b00000000; // Configure PORTB I/O
    TRISC=0b00000000; // Configure PORTC I/O
    TRISD=0b00000000; // Configure PORTD I/O
    TRISE=0b00000000; // Configure PORTE I/O

    ADCON1=0b00001111; // Enable digital I/O

    //Disable all LEDs
    PORTBbits.RB0 = 0;
    PORTBbits.RB1 = 1;
    PORTBbits.RB2 = 1;
    PORTBbits.RB3 = 1;
}

void main( void )
{
    int i;

    init();

    for(i = 0; i<200; i++) {
    MD5Initialize(&hashsum);
    MD5AddData(&hashsum, string, 4);
    MD5Calculate(&hashsum, hash);
```

```
    }

    while(1) {
        PORTBbits.RB0 = 1;
    }
}
```

# B Bijlage 2: hashes.h

```
/*********************************************************************
 *
 *                  Hash Function Library Headers
 *
 *********************************************************************
 * FileName:        Hashes.h
 * Dependencies:    None
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        Microchip C32 v1.05 or higher
 *                    Microchip C30 v3.12 or higher
 *                    Microchip C18 v3.30 or higher
 *                    HI-TECH PICC-18 PRO 9.63PL2 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2009 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c, ENC28J60.h,
 *        ENCX24J600.c and ENCX24J600.h ported to a non-Microchip device
 *        used in conjunction with a Microchip ethernet controller for
 *        the sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
 *
 * IMPORTANT:  The implementation and use of third party algorithms,
 * specifications and/or other technology may require a license from
 * various third parties.  It is your responsibility to obtain
```

```
 * information regarding any applicable licensing obligations.
 *
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Elliott Wood         05/01/07    Original
 ********************************************************************/

#ifndef __HASHES_H
#define __HASHES_H

/*****************************************************************************
  Section:
    Data Types
  *****************************************************************************/

typedef unsigned char           BYTE;                           /* 8-bit unsigned  */
typedef unsigned short int      WORD;                           /* 16-bit unsigned */
typedef unsigned long           DWORD;                          /* 32-bit unsigned */
#define ROM                     rom
#define __EXTENSION
#define __PACKED

typedef union
{
    DWORD Val;
    WORD w[2] __PACKED;
    BYTE v[4] __PACKED;
    struct __PACKED
    {
        WORD LW;
        WORD HW;
    } word;
    struct __PACKED
    {
        BYTE LB;
        BYTE HB;
        BYTE UB;
        BYTE MB;
    } byte;
    /*struct __PACKED
    {
        WORD_VAL low;
        WORD_VAL high;
    }wordUnion;*/
    struct __PACKED
    {
        __EXTENSION BYTE b0:1;
        __EXTENSION BYTE b1:1;
        __EXTENSION BYTE b2:1;
```

```
        __EXTENSION BYTE b3:1;
        __EXTENSION BYTE b4:1;
        __EXTENSION BYTE b5:1;
        __EXTENSION BYTE b6:1;
        __EXTENSION BYTE b7:1;
        __EXTENSION BYTE b8:1;
        __EXTENSION BYTE b9:1;
        __EXTENSION BYTE b10:1;
        __EXTENSION BYTE b11:1;
        __EXTENSION BYTE b12:1;
        __EXTENSION BYTE b13:1;
        __EXTENSION BYTE b14:1;
        __EXTENSION BYTE b15:1;
        __EXTENSION BYTE b16:1;
        __EXTENSION BYTE b17:1;
        __EXTENSION BYTE b18:1;
        __EXTENSION BYTE b19:1;
        __EXTENSION BYTE b20:1;
        __EXTENSION BYTE b21:1;
        __EXTENSION BYTE b22:1;
        __EXTENSION BYTE b23:1;
        __EXTENSION BYTE b24:1;
        __EXTENSION BYTE b25:1;
        __EXTENSION BYTE b26:1;
        __EXTENSION BYTE b27:1;
        __EXTENSION BYTE b28:1;
        __EXTENSION BYTE b29:1;
        __EXTENSION BYTE b30:1;
        __EXTENSION BYTE b31:1;
    } bits;
} DWORD_VAL;

// Type of hash being calculated
typedef enum
{
    HASH_MD5    = 0u,        // MD5 is being calculated
    HASH_SHA1                // SHA-1 is being calculated
} HASH_TYPE;

// Context storage for a hash operation
typedef struct
{
    DWORD h0;                // Hash state h0
    DWORD h1;                // Hash state h1
    DWORD h2;                // Hash state h2
    DWORD h3;                // Hash state h3
    DWORD h4;                // Hash state h4
    DWORD bytesSoFar;        // Total number of bytes hashed so far
    BYTE partialBlock[64];    // Beginning of next 64 byte block
    HASH_TYPE hashType;        // Type of hash being calculated
```

```
} HASH_SUM;

/*****************************************************************************
  Section:
    Function Prototypes
  *****************************************************************************/
DWORD leftRotateDWORD(DWORD val, BYTE bits);

#if defined(STACK_USE_SHA1)
    void SHA1Initialize(HASH_SUM* theSum);
    void SHA1AddData(HASH_SUM* theSum, BYTE* data, WORD len);
    void SHA1Calculate(HASH_SUM* theSum, BYTE* result);
    #if defined(__18CXX)
        void SHA1AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len);
    #else
        // Non-ROM variant for C30 / C32
        #define SHA1AddROMData(a,b,c)    SHA1AddData(a,(BYTE*)b,c)
    #endif
#endif

#if defined(STACK_USE_MD5)
    void MD5Initialize(HASH_SUM* theSum);
    void MD5AddData(HASH_SUM* theSum, BYTE* data, WORD len);
    void MD5Calculate(HASH_SUM* theSum, BYTE* result);
    #if defined(__18CXX)
        void MD5AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len);
    #else
        // Non-ROM variant for C30 / C32
        #define MD5AddROMData(a,b,c)    MD5AddData(a,(BYTE*)b,c)
    #endif
#endif

void HashAddData(HASH_SUM* theSum, BYTE* data, WORD len);
#if defined(__18CXX)
    void HashAddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len);
#else
    // Non-ROM variant for C30 / C32
    #define HashAddROMData(a,b,c)    HashAddData(a,(BYTE*)b,c)
#endif
#endif
```

# C Bijlage 3: hashes.c

```
/*********************************************************************
 *
 *    Hash Function Library
 *  Library for Microchip TCP/IP Stack
 *     -Calculates MD5 and SHA-1 Hashes
 *     -Reference: RFC 1321 (MD5), RFC 3174 and FIPS 180-1 (SHA-1)
 *
 *********************************************************************
 * FileName:        Hashes.c
 * Dependencies:    None
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        Microchip C32 v1.05 or higher
 *                    Microchip C30 v3.12 or higher
 *                    Microchip C18 v3.30 or higher
 *                    HI-TECH PICC-18 PRO 9.63PL2 or higher
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * Copyright (C) 2002-2009 Microchip Technology Inc.  All rights
 * reserved.
 *
 * Microchip licenses to you the right to use, modify, copy, and
 * distribute:
 * (i)  the Software when embedded on a Microchip microcontroller or
 *      digital signal controller product ("Device") which is
 *      integrated into Licensee's product; or
 * (ii) ONLY the Software driver source files ENC28J60.c, ENC28J60.h,
 *        ENCX24J600.c and ENCX24J600.h ported to a non-Microchip device
 *        used in conjunction with a Microchip ethernet controller for
 *        the sole purpose of interfacing with the ethernet controller.
 *
 * You should refer to the license agreement accompanying this
 * Software for additional information regarding your rights and
 * obligations.
 *
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL
 * MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF
 * PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
 * BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
 * THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER
 * SIMILAR COSTS, WHETHER ASSERTED ON THE BASIS OF CONTRACT, TORT
 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE.
```

```
 *
 * IMPORTANT:  The implementation and use of third party algorithms,
 * specifications and/or other technology may require a license from
 * various third parties.  It is your responsibility to obtain
 * information regarding any applicable licensing obligations.
 *
 *
 * Author               Date        Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Elliott Wood            5/01/07        Original
 * Elliott Wood            11/21/07    Greatly increased HashBlock speed
 ********************************************************************/
#define __HASHES_C
#define STACK_USE_MD5

/*****************************************************************************
  Internal:
      Performance Statistics are as follows:
      (Given in instructions per block = 512 bits = 64 bytes)

                    MD5                     SHA1
        C18            23k instr/block        50k instr/block
        Hi-Tech C    19k instr/block        50k instr/block
        C30            21k instr/block        17k instr/block

  *****************************************************************************/

//#include "TCPIP Stack/TCPIP.h"
#include <string.h>
#include "Hashes.h"

/*****************************************************************************
  Section:
    Functions and variables required for both hash types
  *****************************************************************************/

#if defined(STACK_USE_MD5) || defined(STACK_USE_SHA1)

// Stores a copy of the last block with the required padding
BYTE lastBlock[64];

/*****************************************************************************
  Function:
    void HashAddData(HASH_SUM* theSum, BYTE* data, WORD len)

  Description:
    Adds data to the hash sum.

  Precondition:
    The hash sum has already been initialized
```

```
  Parameters:
    theSum - hash context state
    data - the data to be added to the hash sum
    len - length of data

  Returns:
      None

  Remarks:
    This function calls the appropriate hashing function based on the
    hash typed defined in theSum.
  ***************************************************************************/
void HashAddData(HASH_SUM* theSum, BYTE* data, WORD len)
{
    #if defined(STACK_USE_MD5)
    if(theSum->hashType == HASH_MD5)
        MD5AddData(theSum, data, len);
    #endif
    #if defined(STACK_USE_SHA1)
    if(theSum->hashType == HASH_SHA1)
        SHA1AddData(theSum, data, len);
    #endif
}

/*****************************************************************************
  Function:
    void HashAddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)

  Description:
    Adds data to the hash sum.

  Precondition:
    The hash sum has already been initialized

  Parameters:
    theSum - hash context state
    data - the data to be added to the hash sum
    len - length of data

  Returns:
      None

  Remarks:
    This function calls the appropriate hashing function based on the
    hash typed defined in theSum.

    This function is aliased to HashAddData on non-PIC18 platforms.
  ***************************************************************************/
#if defined(__18CXX)
```

```
void HashAddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)
{
    #if defined(STACK_USE_MD5)
    if(theSum->hashType == HASH_MD5)
        MD5AddROMData(theSum, data, len);
    #endif
    #if defined(STACK_USE_SHA1)
    if(theSum->hashType == HASH_SHA1)
        SHA1AddROMData(theSum, data, len);
    #endif
}
#endif

#endif

/*****************************************************************************
  Section:
    Functions and variables required for MD5
  *****************************************************************************/

#if defined(STACK_USE_MD5)

// Array of pre-defined R vales for MD5
static ROM BYTE _MD5_r[64] = {
    7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,
    5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,
    4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,
    6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21};

// Array of pre-defined K values for MD5
static ROM DWORD _MD5_k[64] = {
    0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE,
    0xF57C0FAF, 0x4787C62A, 0xA8304613, 0xFD469501,
    0x698098D8, 0x8B44F7AF, 0xFFFF5BB1, 0x895CD7BE,
    0x6B901122, 0xFD987193, 0xA679438E, 0x49B40821,
    0xF61E2562, 0xC040B340, 0x265E5A51, 0xE9B6C7AA,
    0xD62F105D, 0x02441453, 0xD8A1E681, 0xE7D3FBC8,
    0x21E1CDE6, 0xC33707D6, 0xF4D50D87, 0x455A14ED,
    0xA9E3E905, 0xFCEFA3F8, 0x676F02D9, 0x8D2A4C8A,
    0xFFFA3942, 0x8771F681, 0x6D9D6122, 0xFDE5380C,
    0xA4BEEA44, 0x4BDECFA9, 0xF6BB4B60, 0xBEBFBC70,
    0x289B7EC6, 0xEAA127FA, 0xD4EF3085, 0x04881D05,
    0xD9D4D039, 0xE6DB99E5, 0x1FA27CF8, 0xC4AC5665,
    0xF4292244, 0x432AFF97, 0xAB9423A7, 0xFC93A039,
    0x655B59C3, 0x8F0CCC92, 0xFFEFF47D, 0x85845DD1,
    0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1,
    0xF7537E82, 0xBD3AF235, 0x2AD7D2BB, 0xEB86D391
};

static void MD5HashBlock(BYTE* data, DWORD* h0, DWORD* h1, DWORD* h2, DWORD* h3);
```

```
/*****************************************************************************
  Function:
    void MD5Initialize(HASH_SUM* theSum)

  Description:
    Initializes a new MD5 hash.

  Precondition:
    None

  Parameters:
    theSum - pointer to the allocated HASH_SUM object to initialize as MD5

  Returns:
      None
  *****************************************************************************/
void MD5Initialize(HASH_SUM* theSum)
{
    theSum->h0 = 0x67452301;
    theSum->h1 = 0xefcdab89;
    theSum->h2 = 0x98badcfe;
    theSum->h3 = 0x10325476;
    theSum->bytesSoFar = 0;
    theSum->hashType = HASH_MD5;
}

/*****************************************************************************
  Function:
    void MD5AddData(HASH_SUM* theSum, BYTE* data, WORD len)

  Description:
    Adds data to an MD5 hash calculation.

  Precondition:
    The hash context has already been initialized.

  Parameters:
    theSum - a pointer to the hash context structure
    data - the data to add to the hash
    len - the length of the data to add

  Returns:
      None
  *****************************************************************************/
void MD5AddData(HASH_SUM* theSum, BYTE* data, WORD len)
{
    BYTE *blockPtr;

    // Seek to the first free byte
```

```
    blockPtr = theSum->partialBlock + ( theSum->bytesSoFar & 0x3f );

    // Update the total number of bytes
    theSum->bytesSoFar += len;

    // Copy data into the partial block
    while(len != 0u)
    {
        *blockPtr++ = *data++;

        // If the partial block is full, hash the data and start over
        if(blockPtr == theSum->partialBlock + 64)
        {
            MD5HashBlock(theSum->partialBlock, &theSum->h0,
                         &theSum->h1, &theSum->h2, &theSum->h3);
            blockPtr = theSum->partialBlock;
        }

        len--;
    }
}

/*****************************************************************************
  Function:
    void MD5AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)

  Description:
    Adds data to an MD5 hash calculation.

  Precondition:
    The hash context has already been initialized.

  Parameters:
    theSum - a pointer to the hash context structure
    data - the data to add to the hash
    len - the length of the data to add

  Returns:
      None

  Remarks:
      This function is aliased to MD5AddData on non-PIC18 platforms.
  *****************************************************************************/
#if defined(__18CXX)
void MD5AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)
{
    BYTE *blockPtr;

    // Seek to the first free byte
    blockPtr = theSum->partialBlock + ( theSum->bytesSoFar & 0x3f );
```

```
    // Update the total number of bytes
    theSum->bytesSoFar += len;

    // Copy data into the partial block
    while(len != 0u)
    {
        *blockPtr++ = *data++;

        // If the partial block is full, hash the data and start over
        if(blockPtr == theSum->partialBlock + 64)
        {
            MD5HashBlock(theSum->partialBlock, &theSum->h0,
                         &theSum->h1, &theSum->h2, &theSum->h3);
            blockPtr = theSum->partialBlock;
        }

        len--;
    }
}
#endif

/*****************************************************************************
  Function:
    static void MD5HashBlock(BYTE* data, DWORD* h0, DWORD* h1,
                             DWORD* h2, DWORD* h3)

  Summary:
    Calculates the MD5 hash sum of a block.

  Description:
    This function calculates the MD5 hash sum over a block and updates
    the values of h0-h3 with the next context.

  Precondition:
    The data pointer must be WORD aligned on 16-bit parts and DWORD
    aligned on 32-bit PICs.  If alignment is not correct, a memory alignment
    exception will occur.

  Parameters:
    data - The block of 64 bytes to hash
    h0 - the current hash context h0 value
    h1 - the current hash context h1 value
    h2 - the current hash context h2 value
    h3 - the current hash context h3 value

  Returns:
      None

  Internal:
```

```
      TODO convert data to a DWORD* or read from the pointer using byte
      accesses only to avoid any accidental alignment errors
  ***************************************************************************/
static void MD5HashBlock(BYTE* data, DWORD* h0, DWORD* h1, DWORD* h2, DWORD* h3)
{
    DWORD a, b, c, d, f, temp;
    BYTE i, j;

    // Set up a, b, c, d
    a = *h0;
    b = *h1;
    c = *h2;
    d = *h3;

    // Main mixer loop for 64 operations
    for(i = 0; i < 64u; i++)
    {
        if(i <= 15u)
        {
            f = (b & c) | ((~b) & d);
            j = i;
        }
        else if(i > 15u && i <= 31u)
        {
            f = (d & b) | ((~d) & c);
            j = (5 * i + 1) & 0x0f;
        }
        else if(i > 31u && i <= 47u)
        {
            f = (b ^ c ^ d);
            j = (3 * i + 5) & 0x0f;
        }
        else
        {
            f = c ^ (b | (~d));
            j = (7 * i) & 0x0f;
        }

        // Calculate the new mixer values
        temp = d;
        d = c;
        c = b;
        j *= 4;
        b = leftRotateDWORD(a+f+_MD5_k[i]+(*(DWORD*)&data[j]),_MD5_r[i]) + b;
        a = temp;
    }

    // Add the new hash to the sum
    *h0 += a;
    *h1 += b;
```

```
    *h2 += c;
    *h3 += d;

}

/*********************************************************************
  Function:
    void MD5Calculate(HASH_SUM* theSum, BYTE* result)

  Summary:
    Calculates an MD5 hash

  Description:
    This function calculates the hash sum of all input data so far.  It is
    non-destructive to the hash context, so more data may be added after
    this function is called.

  Precondition:
    The hash context has been properly initialized.

  Parameters:
    theSum - the current hash context
    result - 16 byte array in which to store the resulting hash

  Returns:
      None
  *********************************************************************/
void MD5Calculate(HASH_SUM* theSum, BYTE* result)
{
    DWORD h0, h1, h2, h3;
    BYTE i, *partPtr, *endPtr;

    // Initialize the hash variables
    h0 = theSum->h0;
    h1 = theSum->h1;
    h2 = theSum->h2;
    h3 = theSum->h3;

    // Find out how far along we are in the partial block and copy to last block
    partPtr = theSum->partialBlock;
    endPtr = partPtr + ( theSum->bytesSoFar & 0x3f );
    for(i = 0; partPtr != endPtr; i++)
    {
        lastBlock[i] = *partPtr++;
    }

    // Add one more 1 bit and 7 zeros
    lastBlock[i++] = 0x80;

    // If there's 8 or more bytes left to 64, then this is the last block
```

```
    if(i > 56u)
    {// If there's not enough space, then zero fill this and add a new block
        // Zero pad the remainder
        for( ; i < 64u; lastBlock[i++] = 0x00);

        // Calculate a hash on this block and add it to the sum
        MD5HashBlock(lastBlock, &h0, &h1, &h2, &h3);

        // Create a new block for the size
        i = 0;
    }

    // Zero fill the rest of the block
    for( ; i < 56u; lastBlock[i++] = 0x00);

    // Fill in the size, in bits, in little-endian
    lastBlock[56] = theSum->bytesSoFar << 3;
    lastBlock[57] = theSum->bytesSoFar >> 5;
    lastBlock[58] = theSum->bytesSoFar >> 13;
    lastBlock[59] = theSum->bytesSoFar >> 21;
    lastBlock[60] = theSum->bytesSoFar >> 29;
    lastBlock[61] = 0;
    lastBlock[62] = 0;
    lastBlock[63] = 0;

    // Calculate a hash on this final block and add it to the sum
    MD5HashBlock(lastBlock, &h0, &h1, &h2, &h3);

    // Format the result in little-endian format
    memcpy((void*)result,(void*)&h0,4);
    memcpy((void*)&result[4],(void*)&h1,4);
    memcpy((void*)&result[8],(void*)&h2,4);
    memcpy((void*)&result[12],(void*)&h3,4);
}

#endif //ends MD5

/*****************************************************************************
  Section:
    Functions and variables required for SHA-1
  *****************************************************************************/

#if defined(STACK_USE_SHA1)

static void SHA1HashBlock(BYTE* data, DWORD* h0,
                          DWORD* h1, DWORD* h2, DWORD* h3, DWORD* h4);

/*****************************************************************************
  Function:
    void SHA1Initialize(HASH_SUM* theSum)
```

```
  Description:
    Initializes a new SHA-1 hash.

  Precondition:
    None

  Parameters:
    theSum - pointer to the allocated HASH_SUM object to initialize as SHA-1

  Returns:
      None
  ***************************************************************************/
void SHA1Initialize(HASH_SUM* theSum)
{
    theSum->h0 = 0x67452301;
    theSum->h1 = 0xEFCDAB89;
    theSum->h2 = 0x98BADCFE;
    theSum->h3 = 0x10325476;
    theSum->h4 = 0xC3D2E1F0;
    theSum->bytesSoFar = 0;
    theSum->hashType = HASH_SHA1;
}

/***************************************************************************
  Function:
    void SHA1AddData(HASH_SUM* theSum, BYTE* data, WORD len)

  Description:
    Adds data to a SHA-1 hash calculation.

  Precondition:
    The hash context has already been initialized.

  Parameters:
    theSum - a pointer to the hash context structure
    data - the data to add to the hash
    len - the length of the data to add

  Returns:
      None
  ***************************************************************************/
void SHA1AddData(HASH_SUM* theSum, BYTE* data, WORD len)
{
    BYTE *blockPtr;

    // Seek to the first free byte
    blockPtr = theSum->partialBlock + ( theSum->bytesSoFar & 0x3f );

    // Update the total number of bytes
```

```
    theSum->bytesSoFar += len;

    // Copy data into the partial block
    while(len != 0u)
    {
        *blockPtr++ = *data++;

        // If the partial block is full, hash the data and start over
        if(blockPtr == theSum->partialBlock + 64)
        {
            SHA1HashBlock(theSum->partialBlock, &theSum->h0, &theSum->h1,
                          &theSum->h2, &theSum->h3, &theSum->h4);
            blockPtr = theSum->partialBlock;
        }

        len--;
    }

}

/*****************************************************************************
  Function:
    void SHA1AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)

  Description:
    Adds data to a SHA-1 hash calculation.

  Precondition:
    The hash context has already been initialized.

  Parameters:
    theSum - a pointer to the hash context structure
    data - the data to add to the hash
    len - the length of the data to add

  Returns:
      None

  Remarks:
      This function is aliased to SHA1AddData on non-PIC18 platforms.
  *****************************************************************************/
#if defined(__18CXX)
void SHA1AddROMData(HASH_SUM* theSum, ROM BYTE* data, WORD len)
{
    BYTE *blockPtr;

    // Seek to the first free byte
    blockPtr = theSum->partialBlock + ( theSum->bytesSoFar & 0x3f );

    // Update the total number of bytes
```

```
    theSum->bytesSoFar += len;

    // Copy data into the partial block
    while(len != 0u)
    {
        *blockPtr++ = *data++;

        // If the partial block is full, hash the data and start over
        if(blockPtr == theSum->partialBlock + 64)
        {
            SHA1HashBlock(theSum->partialBlock, &theSum->h0, &theSum->h1,
                          &theSum->h2, &theSum->h3, &theSum->h4);
            blockPtr = theSum->partialBlock;
        }

        len--;
    }

}
#endif

/*****************************************************************************
  Function:
    static void SHA1HashBlock(BYTE* data, DWORD* h0, DWORD* h1,
                              DWORD* h2, DWORD* h3, DWORD* h4)

  Summary:
    Calculates the SHA-1 hash sum of a block.

  Description:
    This function calculates the SHA-1 hash sum over a block and updates
    the values of h0-h3 with the next context.

  Precondition:
    The data pointer must be WORD aligned on 16-bit parts and DWORD
    aligned on 32-bit PICs.  If alignment is not correct, a memory alignment
    exception will occur.

  Parameters:
    data - The block of 64 bytes to hash
    h0 - the current hash context h0 value
    h1 - the current hash context h1 value
    h2 - the current hash context h2 value
    h3 - the current hash context h3 value
    h4 - the current hash context h4 value

  Returns:
      None

  Internal:
```

```
      TODO convert data to a DWORD* or read from the pointer using byte
      accesses only to avoid any accidental alignment errors
  ********************************************************************/
static void SHA1HashBlock(BYTE* data, DWORD* h0, DWORD* h1, DWORD* h2,
                          DWORD* h3, DWORD* h4)
{
    DWORD a, b, c, d, e, f, k, temp;
    DWORD_VAL *w = (DWORD_VAL*)lastBlock;
    BYTE i, back3, back8, back14;

    // Set up a, b, c, d, e
    a = *h0;
    b = *h1;
    c = *h2;
    d = *h3;
    e = *h4;

    // Set up the w[] vector
    if(lastBlock == data)
    {// If they're the same, just swap endian-ness
        for(i = 0; i < 16u; i++)
        {
            back3 = data[3];
            data[3] = data[0];
            data[0] = back3;
            back3 = data[1];
            data[1] = data[2];
            data[2] = back3;
            data += 4;
        }
    }
    else
    {// Otherwise, copy values in swaping endian-ness as we go
        for(i = 0; i < 16u; i++)
        {
            w[i].v[3] = *data++;
            w[i].v[2] = *data++;
            w[i].v[1] = *data++;
            w[i].v[0] = *data++;
        }
    }
    back3 = 13;
    back8 = 8;
    back14 = 2;

    // Main mixer loop for 80 operations
    for(i = 0; i < 80u; i++)
    {
        if(i <= 19u)
        {
```

```
    f = (b & c) | ((~b) & d);
    k = 0x5A827999;
}
else if(i >= 20u && i <= 39u)
{
    f = b ^ c ^ d;
    k = 0x6ED9EBA1;
}
else if(i >= 40u && i <= 59u)
{
    f = (b & c) | (b & d) | (c & d);
    k = 0x8F1BBCDC;
}
else
{
    f = b ^ c ^ d;
    k = 0xCA62C1D6;
}

// Calculate the w[] value and store it in the array for future use
if(i >= 16u)
{
    #if defined(HI_TECH_C)
    // This section is unrolled for HI_TECH_C because it cannot parse
    // the expression used by the other compilers
    DWORD temp2;
    temp = w[back3].Val;
    temp2 = w[back8].Val;
    temp ^= temp2;
    temp2 = w[back14].Val;
    temp ^= temp2;
    temp2 = w[i&0x0f].Val;
    temp ^= temp2;
    w[i&0x0f].Val = leftRotateDWORD(temp, 1);
    #else
    w[i&0x0f].Val = leftRotateDWORD( ( w[back3].Val ^ w[back8].Val ^
                                       w[back14].Val ^ w[i&0x0f].Val), 1);
    #endif
    back3 += 1;
    back8 += 1;
    back14 += 1;
    back3 &= 0x0f;
    back8 &= 0x0f;
    back14 &= 0x0f;
}

// Calculate the new mixers
temp = leftRotateDWORD(a, 5) + f + e + k + w[i & 0x0f].Val;
e = d;
d = c;
```

```
        c = leftRotateDWORD(b, 30);
        b = a;
        a = temp;
    }

    // Add the new hash to the sum
    *h0 += a;
    *h1 += b;
    *h2 += c;
    *h3 += d;
    *h4 += e;

}

/*****************************************************************************
  Function:
    void MD5Calculate(HASH_SUM* theSum, BYTE* result)

  Summary:
    Calculates a SHA-1 hash

  Description:
    This function calculates the hash sum of all input data so far.  It is
    non-destructive to the hash context, so more data may be added after
    this function is called.

  Precondition:
    The hash context has been properly initialized.

  Parameters:
    theSum - the current hash context
    result - 20 byte array in which to store the resulting hash

  Returns:
      None
  *****************************************************************************/
void SHA1Calculate(HASH_SUM* theSum, BYTE* result)
{
    DWORD h0, h1, h2, h3, h4;
    BYTE i, *partPtr, *endPtr;

    // Initialize the hash variables
    h0 = theSum->h0;
    h1 = theSum->h1;
    h2 = theSum->h2;
    h3 = theSum->h3;
    h4 = theSum->h4;

    // Find out how far along we are in the partial block and copy to last block
    partPtr = theSum->partialBlock;
```

```
endPtr = partPtr + ( theSum->bytesSoFar & 0x3f );
for(i = 0; partPtr != endPtr; i++)
{
    lastBlock[i] = *partPtr++;
}

// Add one more bit and 7 zeros
lastBlock[i++] = 0x80;

// If there's 8 or more bytes left to 64, then this is the last block
if(i > 56u)
{// If there's not enough space, then zero fill this and add a new block
    // Zero pad the remainder
    for( ; i < 64u; lastBlock[i++] = 0x00);

    // Calculate a hash on this block and add it to the sum
    SHA1HashBlock(lastBlock, &h0, &h1, &h2, &h3, &h4);

    //create a new block for the size
    i = 0;
}

// Zero fill the rest of the block
for( ; i < 56u; lastBlock[i++] = 0x00);

// Fill in the size, in bits, in big-endian
lastBlock[63] = theSum->bytesSoFar << 3;
lastBlock[62] = theSum->bytesSoFar >> 5;
lastBlock[61] = theSum->bytesSoFar >> 13;
lastBlock[60] = theSum->bytesSoFar >> 21;
lastBlock[59] = theSum->bytesSoFar >> 29;
lastBlock[58] = 0;
lastBlock[57] = 0;
lastBlock[56] = 0;

// Calculate a hash on this final block and add it to the sum
SHA1HashBlock(lastBlock, &h0, &h1, &h2, &h3, &h4);

// Format the result in big-endian format
*result++ = ((BYTE*)&h0)[3];
*result++ = ((BYTE*)&h0)[2];
*result++ = ((BYTE*)&h0)[1];
*result++ = ((BYTE*)&h0)[0];
*result++ = ((BYTE*)&h1)[3];
*result++ = ((BYTE*)&h1)[2];
*result++ = ((BYTE*)&h1)[1];
*result++ = ((BYTE*)&h1)[0];
*result++ = ((BYTE*)&h2)[3];
*result++ = ((BYTE*)&h2)[2];
*result++ = ((BYTE*)&h2)[1];
```

```
    *result++ = ((BYTE*)&h2)[0];
    *result++ = ((BYTE*)&h3)[3];
    *result++ = ((BYTE*)&h3)[2];
    *result++ = ((BYTE*)&h3)[1];
    *result++ = ((BYTE*)&h3)[0];
    *result++ = ((BYTE*)&h4)[3];
    *result++ = ((BYTE*)&h4)[2];
    *result++ = ((BYTE*)&h4)[1];
    *result++ = ((BYTE*)&h4)[0];
}

#endif

/*****************************************************************************
  Function:
    DWORD leftRotateDWORD(DWORD val, BYTE bits)

  Summary:
    Left-rotates a DWORD.

  Description:
    This function rotates the bits in a 32-bit DWORD left by a specific
    number of bits.

  Precondition:
    None

  Parameters:
    val        - the DWORD to be rotated
    bits    - the number of bits by which to shift

  Returns:
    Rotated DWORD value.

  Remarks:
    This function is only implemented on 8-bit platforms for now.  The
    8-bit compilers generate excessive code for this function, while C30
    and C32 already generate compact code.  Those compilers are served
    by a macro defined in Helpers.h.
  *****************************************************************************/
#if defined(__18CXX)
DWORD leftRotateDWORD(DWORD val, BYTE bits)
{
    BYTE i, t;
    DWORD_VAL toRotate;
    toRotate.Val = val;

    for(i = bits; i >= 8u; i -= 8)
    {
        t = toRotate.v[3];
```

```
        toRotate.v[3] = toRotate.v[2];
        toRotate.v[2] = toRotate.v[1];
        toRotate.v[1] = toRotate.v[0];
        toRotate.v[0] = t;
    }

    #if defined(HI_TECH_C)
    for(; i != 0; i--)
    {
        asm("movlb (_toRotate)>>8");
        //asm("bcf _STATUS,0,C");
        asm("bcf 0xFD8,0,C"); // HI-TECH PICC-18 PRO 9.63PL1 doesn't define _STATUS
        asm("btfsc (_toRotate)&0ffh+3,7,B");
        //asm("bsf _STATUS,0,C");
        asm("bsf 0xFD8,0,C"); // HI-TECH PICC-18 PRO 9.63PL1 doesn't define _STATUS
        asm("rlcf (_toRotate)&0ffh+0,F,B");
        asm("rlcf (_toRotate)&0ffh+1,F,B");
        asm("rlcf (_toRotate)&0ffh+2,F,B");
        asm("rlcf (_toRotate)&0ffh+3,F,B");
    }
    #else
    for(; i != 0u; i--)
    {
        _asm
        movlb toRotate
        bcf STATUS,0,0
        btfsc toRotate+3,7,1
        bsf STATUS,0,0
        rlcf toRotate+0,1,1
        rlcf toRotate+1,1,1
        rlcf toRotate+2,1,1
        rlcf toRotate+3,1,1
        _endasm
    }
    #endif

    return toRotate.Val;
}
#endif
```