AVR231 : AES(新暗号化規格)ﾌﾞｰﾄﾛｰﾀﾞ

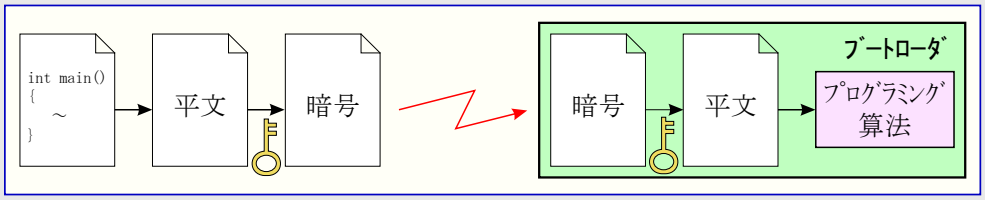
要点

- ■ ﾌﾞｰﾄﾛｰﾀﾞ能力と最低1KﾊﾞｲﾄのSRAMを持つAtmel® AVR®ﾏｲｸﾛ ｺﾝﾄﾛｰﾗに適合
- ■ AVRに基づく応用へﾌｧｰﾑｳｪｱと重要なﾃﾞｰﾀの安全な転送を許す
- ■ 簡単な使用と構成設定可能な応用例を包含
  - ・ 2進ﾌｧｲﾙとﾃﾞｰﾀの暗号化
  - ・ 目的対象ﾌﾞｰﾄﾛｰﾀﾞ作成
  - ・ 目的対象への暗号化ﾌｧｲﾙのﾀﾞｳﾝﾛｰﾄﾞ
- ■ 新暗号化規格(AES)実装
  - ・ 128, 192, 256ﾋﾞｯﾄ鍵
- ■ 2Kﾊﾞｲﾄ内に合うAESﾌﾞｰﾄﾛｰﾀﾞ
- ■ 64Kﾊﾞｲﾄ応用、115200ﾎﾞｰ、3.69MHz目的対象周波数での代表的な更新時間
  - ・ AES128 : 27秒
  - ・ AES192 : 30秒
  - ・ AES256 : 33秒

1. 序説

この応用記述はﾌﾞｰﾄﾛｰﾀﾞ能力を持つAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗでﾌｧｰﾑｳｪｱがどう安全に更新し得るかを記述します。その方法はﾌｧｰﾑｳｪｱを暗号化するのに新暗号化規格(AES)を使います。

図1-1. 概要



ﾏｲｸﾛ ｺﾝﾄﾛｰﾗを含む電気設計は可搬型音楽再生機、ﾍｱﾄﾞﾗｲﾔ、ﾐｼﾝなどではﾌｧｰﾑｳｪｱを装備されることが常に必要です。多くの電気設計が急速に発達するため、既に出荷または販売されてしまった製品を更新することができる必要が増大しています。特に製品が既に最終顧客へ届いてしまっている場合、ﾊｰﾄﾞｳｪｱに変更を行うのは難しくなると言ってよいでしょうが、しかしAVRのようなﾌﾗｯｼｭ ﾏｲｸﾛ ｺﾝﾄﾛｰﾗに基づく製品でﾌｧｰﾑ ｳｪｱは容易に更新することができます。

多くのAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗは要求があり次第、ﾌｧｰﾑｳｪｱ更新を受け取ってﾌﾗｯｼｭ ﾒﾓﾘを書き換えることができるﾌﾞｰﾄﾛｰﾀﾞを作ることが可能なように構成設定されます。ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ空間はﾌﾞｰﾄﾛｰﾀﾞ領域(BLS)と応用領域の2つの領域に分けられます。両領域はﾌﾞｰﾄﾛｰﾀﾞのｺｰﾄﾞがBLSで保護され得ると同時に未だ応用領域でｺｰﾄﾞを更新できるような読み書き保護用の専用施錠ﾋﾞｯﾄを持ちます。従って、BLSの更新方法は外側のｱｸｾｽに備えて簡単に保護することができます。

ﾌｧｰﾑｳｪｱには問題が残っており、それは代表的に、ﾌﾗｯｼｭ ﾒﾓﾘに書かれて施錠ﾋﾞｯﾄが設定される前に保護されないことです。これはﾌｧｰﾑｳｪｱが現場で更新されることが必要な場合にそれが書き込み台または製造業者の建物を出る瞬間から不正ｱｸｾｽに関して開いていることを意味します。

この応用記述はﾌﾗｯｼｭ ﾒﾓﾘとEEPROMへ転送されるべきﾃﾞｰﾀが暗号法を用いることによって何時でも如何にして保護され得るかを示します。この考えは書き込み台を去る前にﾃﾞｰﾀを暗号化して目的対象AVRにﾀﾞｳﾝﾛｰﾄﾞされた後でだけそれを解読することです。この手順はﾌｧｰﾑｳｪｱの不正な複製を防げませんが、暗号化した情報は正しい解読鍵なしでは実質的に無効です。解読鍵は書き込み環境の外側、目的対象AVRの内側の1つの位置にだけ格納されます。鍵は暗号化されたﾃﾞｰﾀから再生成することができません。ﾃﾞｰﾀへのｱｸｾｽを得る唯一の方法は正しい鍵を使うことによるだけです。

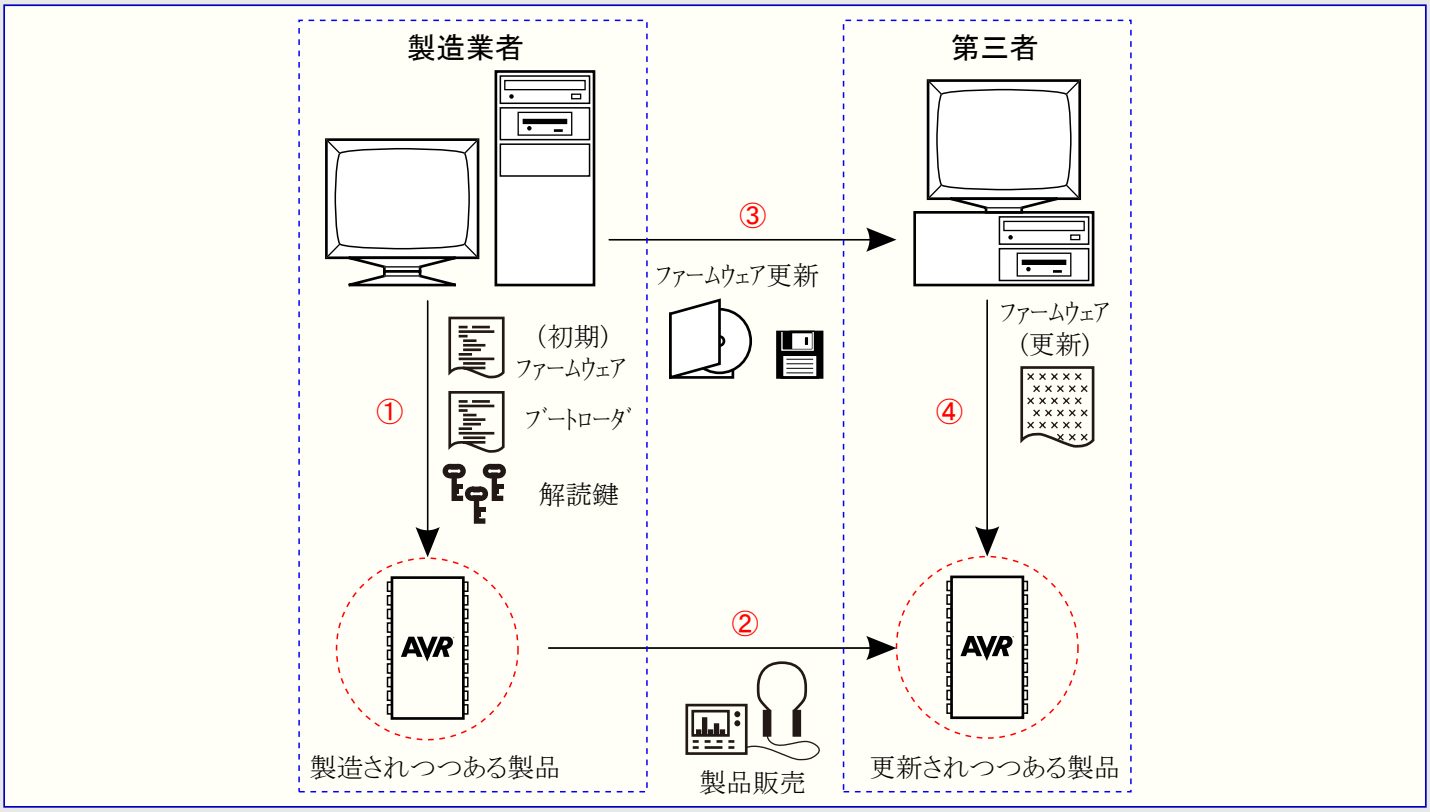
8ﾋﾞｯﾄ Atmel
ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ

応用記述

Rev. 2589E-03/12, 2589EJ2-01/23

図1-2.は製品が最初にどのように製造され、初期ファームウェアを設定され、販売されて、そして後で新版のファームウェアで更新される例を示します。

図1-2. 代表的な製品と更新手順の例



①. 製造中、マイクロ コントローラは最初にブートローダ、解読鍵、応用ファームウェアを装備されます。ブートローダは実際の応用とプログラミングを受け取り、それをフラッシュ メモリ内にプログラミングする準備をし、同時に鍵はやって来るデータの解読に必要です。施錠ビットはAVR内側のファームウェアを保護するように設定されます。

②. 製品はその後に代理店へ出荷されるか、または最終的な顧客に販売されます。施錠ビット設定はAVR内側で保護されたファームウェアを維持し続けます。

③. ファームウェアの新規公開が完了され、既に配給されてしまった製品の更新の必要があります。従って、ファームウェアは暗号化されて代理店に出荷されます。暗号化されたファームウェアは解読鍵なしで用を足さず、従ってソフトウェアの局所的な複製(例えば、代理店のハードドライブ)は安全性を危険に晒しません。

④. 代理店は在庫の全ての構成単位と顧客によって返された(例えば、修理中の)それらを更新します。暗号化されたファームウェアはAVRにダウンロードされ、マイクロ コントローラの内側で一度解読されます。施錠ビット設定はAVRの内側で更新されたファームウェアの保護を維持し続けます。

2. 暗号概要

用語の暗号は情報が鍵を用いて施錠されて利用不能にされる時に用いられます。開錠情報は正しい鍵の使用でだけ達成することができます。

暗号鍵に基づく算法は対称と非対称の2つに分けられます。対称法は暗号化と解読に対して同じ鍵を用い、一方非対称法は異なる鍵を使います。AESは対称鍵法です。

2.1. 暗号化

暗号化はそれの内容が部外者から隠されるようにメッセージやデータを符号化する方法です。その元の形式に於いて平文のメッセージやデータはマイクロ コントローラのファームウェアのように作者や配給者が秘密維持を望む情報を含むかもしれません。例えば、マイクロ コントローラが現場で更新される時に、不正な複製の試みや逆行分析に対してファームウェアを保護することが難しくなるかもしれません。ファームウェアの暗号化はそれが解読されるまでそれを使用できなくします。

2.2. 解読

解読は元のメッセージやデータを取り戻す方法で、代表的に正しい鍵を知ることなく実行することができません。鍵はデバイスが暗号化したデータを受け取ってそれを解読し、フラッシュ メモリまたはEEPROMの選んだ部分を書き換えることができるように、マイクロ コントローラのブートローダ内に格納することができます。解読鍵は暗号化したデータから取り戻すことができず、それ故に施錠ビットがプログラム(0)されてしまっている場合、AVRマイクロ コントローラから読むことができません

3. AES実装

本項はAES算法やその歴史の詳細な記述を意図していません。むしろその意図はこの算法の様々な部分に関してAVR特有実装を記述することです。組み込み応用でﾒﾓﾘは不足しがちな資源なので、焦点はｺｰﾄﾞ ﾒﾓﾘの節約にあります。ﾌﾞｰﾄﾛｰﾀﾞ応用は決して主ｺｰﾄﾞと同時に走行せず、従ってﾃﾞｰﾀ ﾒﾓﾘ必要条件がﾏｲｸﾛ ｺﾝﾄﾛｰﾗの容量を超えない限り、ﾃﾞｰﾀ ﾒﾓﾘ(SRAM)の節約は重要ではありません。

AES実装それ自体に興味がない場合、その読者は継続性を失うことなく7頁の「4. ｿﾌﾄｳｪｱ実装と使い方」章へ直ぐ飛ばすことができます。

後続する副項に於いて、いくつかの基本算術演算とそれらのAVR特有実装が記述されます。数学から有限場理論へいくつかの参照があることに注意してください。この資料を読むのに有限場の知識は必要ありませんが、興味を持った読者はAES仕様を学ぶべきです。

3.1. ﾊﾞｲﾄ加算

AES算法に於いて、ﾊﾞｲﾄ加算はｷｬﾘｰ伝播なしの個別ﾋﾞｯﾄの加算として定義されます。これは一般的なXOR操作が意図されます。XOR操作はそれ自身の逆、従ってﾊﾞｲﾄ減算はAES算法での加算と同等です。XOR操作はAVRで実装するのに取るに足らない問題です。

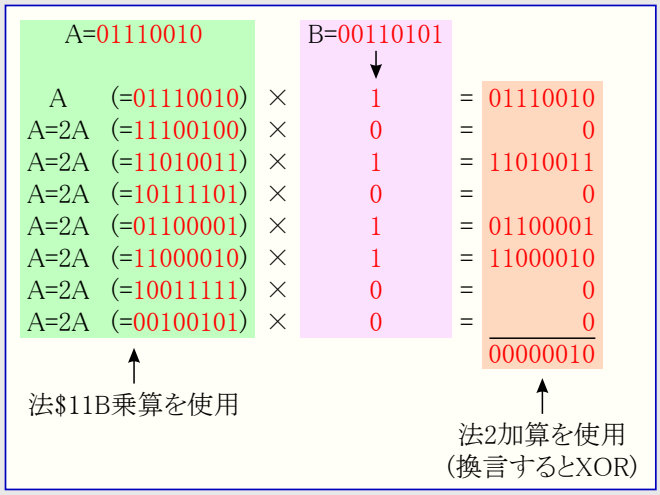
3.2. ﾊﾞｲﾄ乗算

AES算法に於いて、ﾊﾞｲﾄ乗算は法$11B(2進数:1 0001 1011)での有限場乗算として定義されます。提案された実装は最初の因数を2で繰り返し乗算(法$11B)して、値1を持つ2つ目の因数の各ﾋﾞｯﾄに対する中間結果を積和します。例:2つ目の因数が$1A(2進数: 0001 1010)の場合、その後に最初、3つ目、5つ目の中間結果が積和されるべきです。別の例が図3-1.で示されます。この方法は少しのﾒﾓﾘしか使わず、8ﾋﾞｯﾄ ﾏｲｸﾛ ｺﾝﾄﾛｰﾗに上手く適合します。

乗算算法は以下の擬似ｺｰﾄﾞによって記述することができます。

```
bitmask = 1
tempresult = 0
tempfactor = firstfactor
while bitmask < 0x100
    if bitmask AND secondfactor <> 0
        add tempfactor to tempresult using XOR
    end if
    shift bitmask left once
    multiply tempfactor by 2 modulo 0x11B
    end while
return tempresult
```

図3-1. ﾊﾞｲﾄ乗算



3.3. 乗法の逆元

有限場乗法の逆元、換言すると1/xを計算し得るため、この実装ではごまかしが使われます。共通の低で指数と対数を用いて、以下の恒等式を利用することができます。

式3-1. 1/xを計算するための指数と対数の使用

$$a^{-log_a x} = \frac{1}{x}$$

それが分解できない最も基本の根であるため、この場合は低の値は3が選ばれています。指数と対数を計算する時に有限場乗算を用いることにより、乗法の逆元は実装が容易です。毎回、指数と対数を計算する代わりに、2つの参照表が使われます。乗法の逆元は3.4.項で記述されるSﾎﾞｯｸｽを準備する時にだけ使われるので、2つの参照表に使われるﾒﾓﾘはSﾎﾞｯｸｽが準備されてしまっている時に他の目的に使うことができます。

参照表計算は以下の擬似ｺｰﾄﾞによって記述することができます。

```
tempexp = 0
tempnum = 1
do
    exponentiation_table[ tempexp ] = tempnum
    logarithm_table[ tempnum ] = tempexp
    increase tempexp
    multiply tempnum by 3 modulo 0x11B
loop while tempexp < 256
```

3.4. Sﾎﾞｯｸｽ

AES算法は置換表またはSﾎﾞｯｸｽの概念を使います。この算法の段階の1つは可逆変換をﾊﾞｲﾄに適用することです。Sﾎﾞｯｸｽは可能な全てのﾊﾞｲﾄ値に関して予め計算されたこの変換の結果です。この変換は次の2つの段階から成り、(1)3.3.項で記述されるような乗法の逆元と、(2)以下の式に従う線形変換、ここでのa_iは結果のﾋﾞｯﾄ、b_iは段階1からの結果のﾋﾞｯﾄです。

式3-2. Sﾎﾞｯｸｽで使われる線形変換

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

より徹底的な行列の考察は、右手のﾍﾞｸﾄﾙと元のﾊﾞｲﾄの1,2,3,4回の左回転と、元のﾊﾞｲﾄの(XOR加算を用いた)総和として実装することができる操作を明らかにします。この方法は8ﾋﾞｯﾄ ﾏｲｸﾛ ｺﾝﾄﾛｰﾗに上手く適合します。

解読に使われる逆元Sﾎﾞｯｸｽは同様の構造を持ち、また、XOR加算と回転を用いて実装されます。対応する行列についてはAES仕様を、実装の詳細についてはｿｰｽ ｺｰﾄﾞを参照してください。

3.5. '状態'

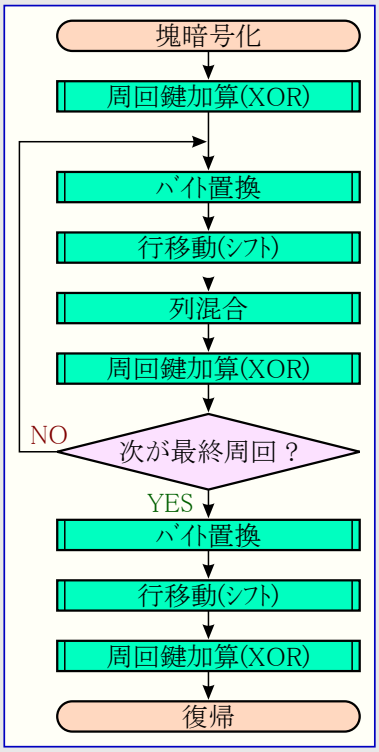
AES算法は塊暗号で、それはﾃﾞｰﾀが塊で管理されることを意味します。AES暗号についてはこの塊容量が16ﾊﾞｲﾄです。このAES塊は度々'状態'または'状態配列'と呼ばれる4×4の配列で構成されます。この状態の最も左側の列は上から下へ塊の最初の4ﾊﾞｲﾄを保持し、以下同様です。読者はAES仕様で4つの連続するﾊﾞｲﾄが語として参照されることにも注意すべきです。

3.6. AES暗号化

暗号化処理の段階を検討する前に、'暗号化周回'の導入が必要です。殆どの塊暗号は何回かの繰り返しで実行される少しの操作から成ります。各繰り返し反復は異なる暗号鍵を用います。各反復内の操作の最低1つでは鍵に依存します。繰り返し反復は暗号化周回として参照され、この周回に使われる鍵の系列が鍵計画と呼ばれます。周回数は鍵の大きさに依存します。

暗号化処理用の流れ図が右の図3-2.で示されます。以下の副項は処理に於ける各種段階を説明します。各段階は利便性のためにｻﾌﾞﾙｰﾁﾝとして実装されます。ｺﾝﾊﾟｲﾗの最適化の使用はｺｰﾄﾞ ﾒﾓﾘを節約するために不要な関数呼び出しを取り去ります。

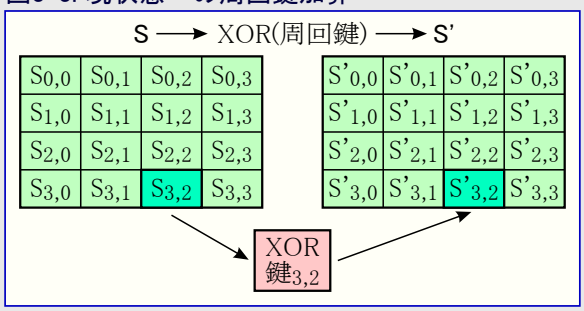
図3-2. 暗号化流れ図



3.6.1. 周回鍵加算

この段階は現在の周回鍵を現在の状態配列に加算するためにXOR加算を使います。周回鍵は状態と同じ大きさ、換言すると16ﾊﾞｲﾄまたは4語を持ちます。この操作は16回の繰り返しとして実装されます。

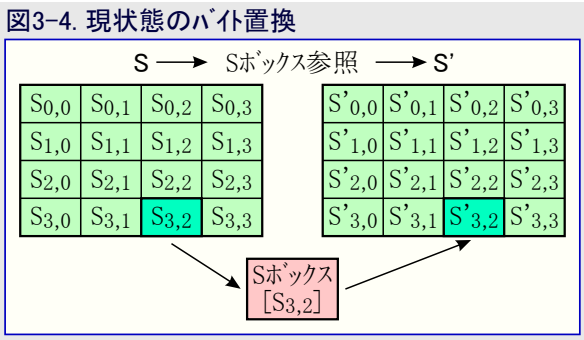
図3-3. 現状態への周回鍵加算



3.6.2. ﾊﾞｲﾄ置換

この段階は状態内でﾊﾞｲﾄを置換するのに予め計算されたSﾎﾞｯｸｽ表を使います。上の3.6.1.項のようにこの段階も16回の繰り返しとして実装されます。

図3-4. 現状態のﾊﾞｲﾄ置換

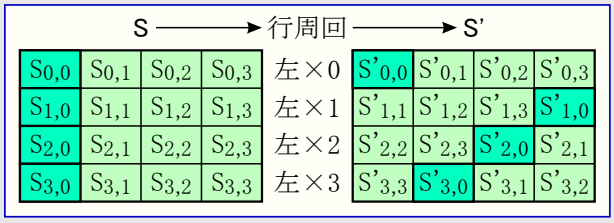


3.6.3. 行移動(ｼﾌﾄ)

この段階は現在の状態の行で操作します。最初の行は元のままで、一方後の3つは各々1、2、3回左に周回されます。1度の左周回には最も左側のﾊﾞｲﾄが最も右側の列に移動され、残りの3つのﾊﾞｲﾄは左に1列移動されます。この処理は図3-5.で示されます。

素直な実装は行を左に1回周回するｻﾌﾞﾙｰﾁﾝを書き、そして各行で必要とされる回数、それを呼ぶことです。けれども、考査はどんな繰り返しやｻﾌﾞﾙｰﾁﾝもなしにﾊﾞｲﾄを直接的に混ぜる実装を示し、ｺｰﾄﾞ量での小さな不利だけなく、速度での重要な利益(3倍)に帰着します。従って直接実行が選ばれます。詳細についてはｿｰｽ ｺｰﾄﾞ内のShiftRows()関数を参照してください。

図3-5. 現状態の行を周回



3.6.4. 列混合

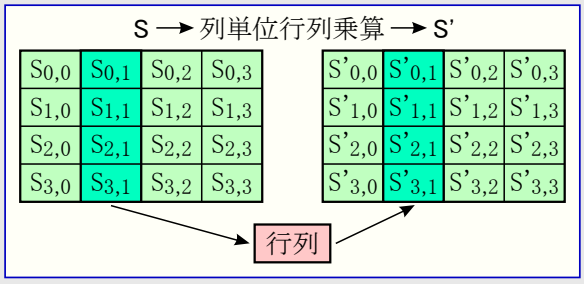
この段階は列単位で状態を操作します。各列はﾊﾞｲﾄのﾍﾞｸﾄﾙとして扱われ、変更した状態に対する列を得るために固定化した行列によって乗算されます。

この操作は以下の式によって記述することができ、ここでのa_iは混合された列のﾊﾞｲﾄ、b_iは元の列のﾊﾞｲﾄです。3.1.項と3.2.項からのXOR加算と有限場乗算が使われることに注意してください。

式3-3. 1列混合時の行列乗算

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 2\ 3\ 1\ 1 \\ 1\ 2\ 3\ 1 \\ 1\ 1\ 2\ 3 \\ 3\ 1\ 1\ 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

図3-6. 現状態の列混合



この段階はどんな二次的関数呼び出しもなしに直接的に実装されます。行列式から混合された列の全てのﾊﾞｲﾄa_iは元のﾊﾞｲﾄb_iとそれらの2倍、$2b_i$の組み組み合わせと見ることができます。詳細についてはｿｰｽ ｺｰﾄﾞ内のMixColumns()関数を参照してください。

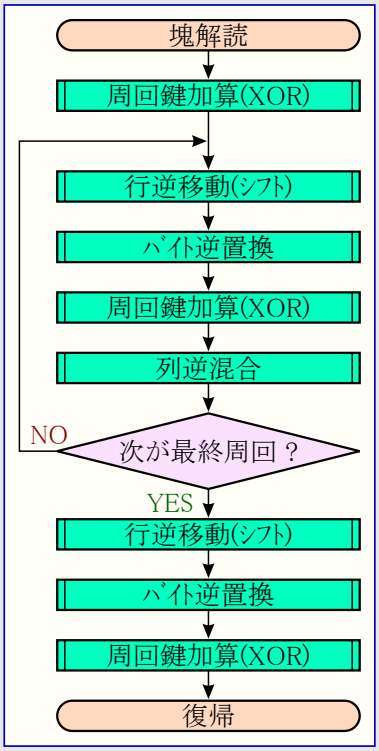
3.7. 解読

解読処理用の流れ図が図3-7.で示されます。この処理は段階の順番が変更されていることを除き、暗号化処理と非常に似ています。"周回鍵加算"を除く全ての段階は対応するそれらの逆元を持ちます。"行逆移動(シフト)"は左の代わりに行を右に周回します。"バイト逆置換"は逆元Sボックスを使います。

"列逆混合"も逆元変換を使います。対応する行列についてはAES仕様を、実装の詳細についてはソースコードを参照してください。

注: 解読に使われる鍵計画は暗号化と同じですが、逆順です。

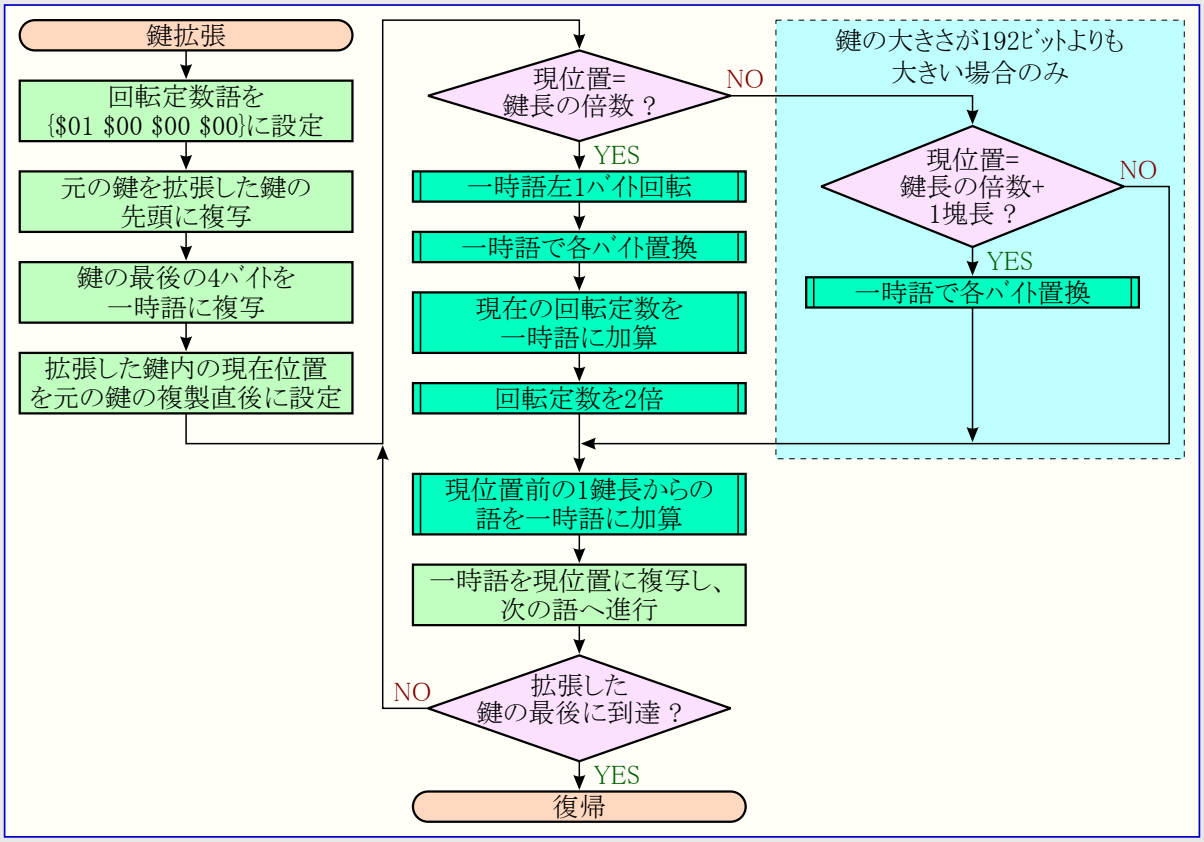
図3-7. 解読流れ図



3.8. 鍵拡張

鍵拡張は元の128,196,256ビット暗号鍵から鍵計画を生成する処理です。鍵拡張用の流れ図が図3-8.で示されます。

図3-8. 鍵拡張流れ図



この算法はXOR加算、有限場乗算、置換、語周回のような既に記述した操作を用います。詳細についてはソース コードを参照してください。

注: 鍵拡張が暗号化と解読の両方に関して同一です。従って例え解読だけが使われる場合でも、暗号化に使われるSボックスが必要とされます。AVR実装では、鍵拡張に先立って通常のSボックスが計算され、それのメモリはその後に逆元Sボックスを計算する時に再利用されます。

3.9. 暗号塊連鎖(CBC:Clipher Block Chaining)

AESは塊暗号で、この算法は固定量のデータ塊で操作することを意味します。データを16バイトの塊に暗号化するのに暗号鍵が使われます。既知の入力塊と一定(例え未知でも)の暗号鍵に関して、その出力は常に同じです。これは暗号システムの攻撃を欲する誰かにとって有用な情報を提供するかもしれません。

同じ平文の塊を違う暗号文の塊に暗号化させるのに一般的に使われる方法があります。そのような方法の1つはが暗号塊連鎖(CBC)と呼ばれます。

CBCは先行塊が全ての後行塊に影響を及ぼすような暗号塊を接続する方法です。現在の平文の塊と直前の暗号文の塊で最初にXOR操作を実行することによって成し遂げられます。XORの結果はその後に平文の塊の代わりに暗号化されます。これは1つの暗号文ビットが依存するところの平文ビット数を増します。

4. ソフトウェア実装と使い方

本章は系の安全性を改善するためのいくつかの重要な項目を最初に検討します。これらの項目は後のソフトウェア設計に於いて多くの決定に動機を与えます。

4.1. 動機付け

この応用記述は外側のアクセスから設計を保護する時に使うことができる技法を提供します。例え完全に保護することができる設計がないとは言え、それは安全性を破るために必要とされる労力が可能な限り高くなるように構成することができます。基本工学的な技能の人が複製することができる安全でない設計と、少数で非常に熟練した侵入者だけが破ることができる設計の間には重大な違いがあります。安全にされていない場合では、設計が容易に複製され、そして製造業者の知的所有権に違反して逆行分析さえもされ、そしてその設計に関する潜在的な市場をも危うくします。安全にされている場合では、設計を破るために必要とされる労力が大きいので、殆どの侵入者は単に彼ら自身の製品を開発することに集中します。

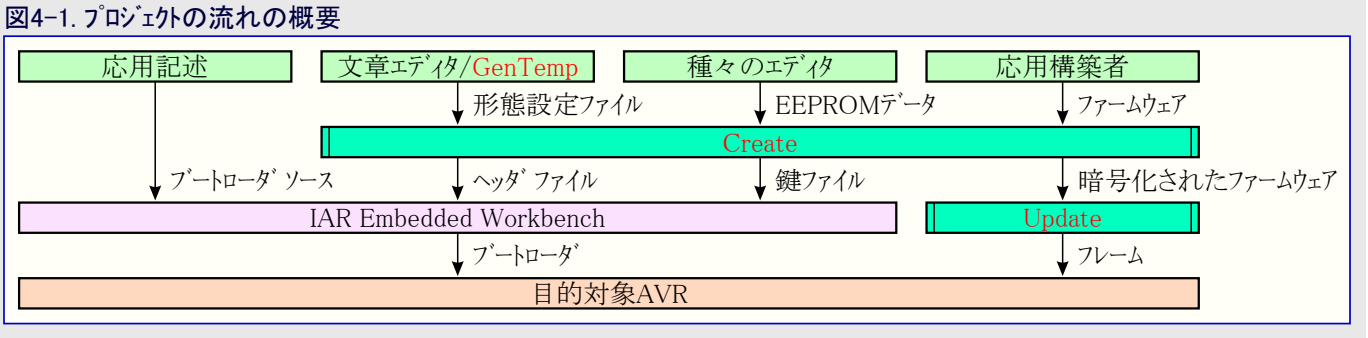
安全なシステム構築法には、可能な限り破るのを難しくするように設計すべきであると言う、一般的な1つの規則だけがあります。破ろうとする間に、保護を欺くのに使うことができるどの手法も試みられるでしょう。考慮されなければならない少しの例が以下で与えられます。

- ファームウェア更新中に電力が取り去られたなら、何が起きますか?。電力が戻って回復した時にマイクロ コントローラの状態はどうですか?。施錠ビットとリセット ベクタは何時も正しく設定されていますか?。
- 平文データのように見える何かを作ることができる何れかの仮定がありますか?。AESに関して総当り法によって破られるには探す様式があるに違いありません。攻撃者は単に全ての鍵の組み合わせを試み、理解できるファームウェアを探してその出力を監視する筈がありません。攻撃ソフトウェアはプログラム メモリの開始での割り込みベクタ、0または1で穴埋めされたメモリ領域、以下同様のような既知の様式を検索するように構成設定されなければなりません。
- 解読処理から得ることができる何れかの還元がありますか?。このようなどの還元も攻撃者を手助けし得ます。例えば、ブートローダ内側の解読算法が各塊処理に対して良/否形式の合図を与える場合、この信号は攻撃者への還元として使われ得ます。
- 符号化したフレームは別の順で送られるべきですか?。ブートローダに送られる最初のフレームが常に暗号化したファイルの最初の塊を含む場合、攻撃者はこれからいくつかの仮定を作ることができます。例えば、最初のフレームはアドレス0から始まるプログラム データの割付でそれは割り込みベクタ表を含むと仮定することができます。この情報は鍵検索を細かく区別する攻撃者を助けます。システムの安全を増すため、乱順でフレームを送ってください(どうせ、解読されたフレームはそれらの正しいアドレスに割付されます)。

4.2. 使い方概要

これと以下のサブルーチンは応用の構成設定と使い方を記述します。この処理は図4-1.で図解されます。

図4-1. プロジェクトの流れの概要



主な段階は次のとおりです。

- 目的対象AVR用の応用を作成してください。必要なら、独立したファイルでEEPROM設計(配置)を作成してください。
- プロジェクト依存情報で構成設定ファイルを作成してください。GenTempと呼ばれる応用がファイル フレーム作成に使えます。
- Createと呼ばれる応用を走らせてください。これはヘッダ ファイル、鍵ファイル、暗号化されたファイルを作成します。
- IAR Embedded Workbench®を使い、目的対象AVR用のブートローダを構成設定して構築してください。
- 目的対象AVRにブートローダをダウンロード(プログラミング)して施錠ビットとヒューズ ビットを設定してください。
- 今や暗号化したファームウェアは何時でもAVRにダウンロード(書くことが)できます。

この手続きは以下でもっと詳細に検討されます。

4.3. 構成設定ファイル

構成設定ファイルはプロジェクトを構成設定するのに使われるパラメータの一覧を含みます。パラメータは表4-1.で記述されます。

表4-1. 構成設定ファイル任意選択の要約

パラメータ	説明	既定	必須
PAGE_SIZE	AVRフラッシュ ページの10進バイトでの大きさ。この値はデバイス依存です、データシートをご覧ください。	N/A	はい
KEY1	16進での暗号鍵の最初の(128ビット)部分。第8ビット後に挿入される奇数パリティ ビットを持つ16の乱数バイトであるべきで、合計18バイトを作成します。	なし: 暗号なし	いいえ、しかし強く推奨
KEY2	16進での暗号鍵の2つ目の(64ビット)部分。第8ビット後に挿入される奇数パリティ ビットを持つ8つの乱数バイトであるべきで、合計9バイトを作成します。省略されたなら、AES128が使われます。	なし: AES128使用	いいえ、しかし推奨
KEY3	16進での暗号鍵の2つ目の(64ビット)部分。第8ビット後に挿入される奇数パリティ ビットを持つ8つの乱数バイトであるべきで、合計9バイトを作成します。省略されたなら、AES128またはAES192が使われます。	なし: AES128またはAES192を使用	いいえ、しかし推奨
INITIAL_VECTOR	暗号塊連鎖に使用。16進での16の乱数バイトであるべきです。	0	いいえ、しかし強く推奨
SIGNATURE	16進でのフレーム確認データ。これはどの4バイトにもできますが、乱数で選ばれた値が推奨されます。	00	いいえ
ENABLE_CRC	CRC検査許可。YesまたはNo。許可されたなら、全ての応用領域は上書きされ、応用はそれが開始を許される前にCRC検査を通らなければなりません。	なし	いいえ、しかし推奨
MEM_SIZE	目的対象AVR内の応用領域の大きさ(10進バイトでの)。	N/A	はい、CRC使用の場合

構成設定ファイルはどの有効なファイル名も与えることができます。この名前はプロジェクト ファイルを作成する応用へのパラメータとして後で与えられます。下はAtmel ATmega16用の試供構成設定ファイルです。KEY1パラメータは挿入されたパリティ ビットを持つ128ビット鍵(16進の$0123456789ABCDEF0123456789ABCDEF)の例です。

```
PAGE_SIZE       = 128
KEY1            = 0111914CE8955B35DE0111914CE8955B35DE
INITIAL_VECTOR  = 00112233445566778899AABBCCDDEEFF
SIGNATURE       = 89ABCDEF
ENABLE_CRC      = YES
MEM_SIZE        = 14336
```

パラメータのいくつかは目的対象AVRの特定の知識なしに設定することができません。表4-2.はブートローダ機能を持つ存在するいくつかのAVRマイクロ コントローラの特性を要約します。この表に存在しないデバイスについてはデバイスのデータシートを参照してください。

表4-2. AVR特性要約

特性	ATmega8	ATmega16	ATmega162	ATmega169	ATmega32	ATmega64	ATmega128
フラッシュ容量(バイト)	8K	16K			32K	64K	128K
フラッシュ ページ容量(バイト)	64	128				256	
フラッシュ ページ数	128				256		512
(最大)BLS容量(バイト)	2048				4096	8192	
BLSページ数	32	16			32		
MEM_SIZE(バイト)	6144	14336			28672	57344	122880
PAGE_SIZE(バイト)	64	128					256

4.4. PC応用 – GenTemp

この応用は構成設定ファイル用の雛形を生成します。この応用は乱数暗号鍵を生成し、ベクタを初期化し、(フラッシュ ページ容量のように)満たされるべき使用者用の他のパラメータをそのままにします。常にこの応用を用いる雛形生成で始めることが推奨されます。

この応用は次のように使われます。

```
GenTemp FileName.Ext
```

ここでFileName.Extは作成されるべき構成設定ファイルの名前です。ファイルが生成された後で選んだ何れかの平文エディタを用いて編集することができます。

4.5. PC応用 – Create

この応用は構成設定ファイルから情報を読んで、ブートローダ用の鍵とヘッダ ファイルを生成します。これはファームウェアの暗号化にも使われます。代表的に、この応用は最低2回走行し、(1)ブートローダ用の鍵とヘッダ ファイルを生成するため、(2)新しいファームウェアが暗号化される時です。

注: プロジェクト ファイルを生成する時とファームウェアを暗号化する時に同じ暗号情報(構成設定ファイル)が使われることが非常に重要です。さもなければ、ブートローダは正しい暗号鍵の組を持たずにデータを解読できないかもしれません。暗号化されたファームウェアを解読するのに構成設定ファイル内の情報を用いることが可能なことにも注意すべきです。従って、形態設定ファイルは何時も安全を保たれなければならず、最初に使われた後で変更されるべきではありません。

4.5.1. 命令行引数

次表は利用可能な命令行引数を示します。

表4-3. 命令行引数の要約

引数	説明
-c <ファイル名.ext>	構成設定ファイルへのパス
-d	設定なら、各フラッシュ ページの内容は書く前に削除されます。そうでなく、特に書かれなければ直前のデータが保たれます。
-e <ファイル名.ext>	EEPROMファイル(EEPROM内容になるデータ)へのパス
-f <ファイル名.ext>	フラッシュ ファイル(応用領域内容になるコード)へのパス
-h <ファイル名.ext>	出力ヘッダ ファイル名。このファイルは後にブートローダでインクルードされなければなりません。
-k <ファイル名.ext>	出力鍵ファイル名。このファイルは後にブートローダでインクルードされなければなりません。
-l [BLB12][BLB11][BLB02][BLB01]	設定する施錠ビット。これらの施錠ビットは全データが転送された後で、制御が更新された応用に渡される前に設定されます。
-n	不感知。暗号化されたファイルに不感知記録の乱数を追加します。不感知記録がブートローダによって無視されるため、この設定は出力ファイルの予測性だけで、応用に影響を及ぼしません。
-o <ファイル名.ext>	出力ファイル名。これは配布され得る暗号化したファイルで、更新が必要な時に目的対象へ送られます。

4.5.2. 初回走行

最初の走行では代表的にブートローダ用の鍵とヘッダ ファイルが生成されるだけです。鍵とヘッダ ファイルの生成は命令行引数を用いて要求されます。

```
例: Create -c Config.txt -h BootLdr.h -k AESKeys.inc
```

鍵とヘッダ ファイルはブートローダ応用のプロジェクト フォルダに複写され、ブートローダ コード内にインクルードされなければなりません。

注: ブートローダ プロジェクト ファイル名は上で言及したファイル名、即ちBootLdr.hとAESKeys.incを使うように、予めコンパイルされています。これらのファイル名は変更されないことが推奨されます。

4.5.3. 後続走行

後続する走行ではﾌｧｰﾑｳｪｱを符号化するのにこの応用が使われます。暗号化に先立ち、ｿｰｽ ﾌｧｲﾙはｺﾝﾊﾟｲﾙ、ｱｾﾝﾌﾞﾙされて1つのｺｰﾄﾞ ｾｸﾞﾒﾝﾄ ﾌｧｲﾙと/または1つのEEPROMｾｸﾞﾒﾝﾄ ﾌｧｲﾙにﾘﾝｸされなければなりません。ﾌｧｲﾙはIntel®のHEX形式でなければなりません。

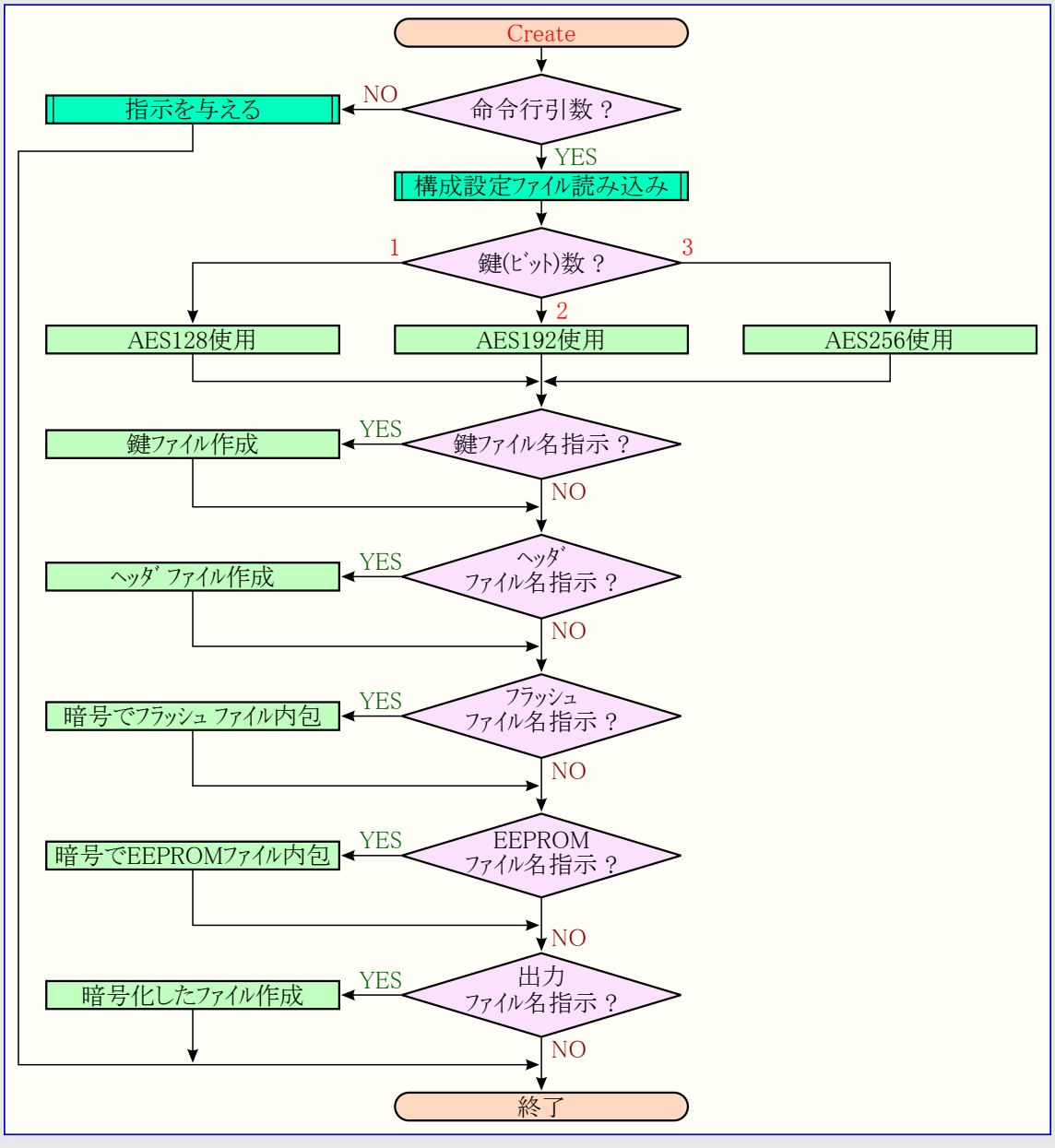
ﾌｧｲﾙ名はｺﾏﾝﾄﾞ ﾌﾟﾛﾝﾌﾟﾄで与えられ、構成設定ﾌｧｲﾙ内のﾃﾞｰﾀに従って暗号化されたﾌｧｲﾙが生成されます。

```
例: Create -c Config.txt -e EEPROM.hex -f Flash.hex -oUpdate.enc -l BLB11 BLB12
```

応用ｿﾌﾄｳｪｱとEEPROMのﾌｧｲﾙは単一の暗号化したﾌｧｲﾙ内に結合されます。

4.5.4. ﾌﾟﾛｸﾞﾗﾑの流れ

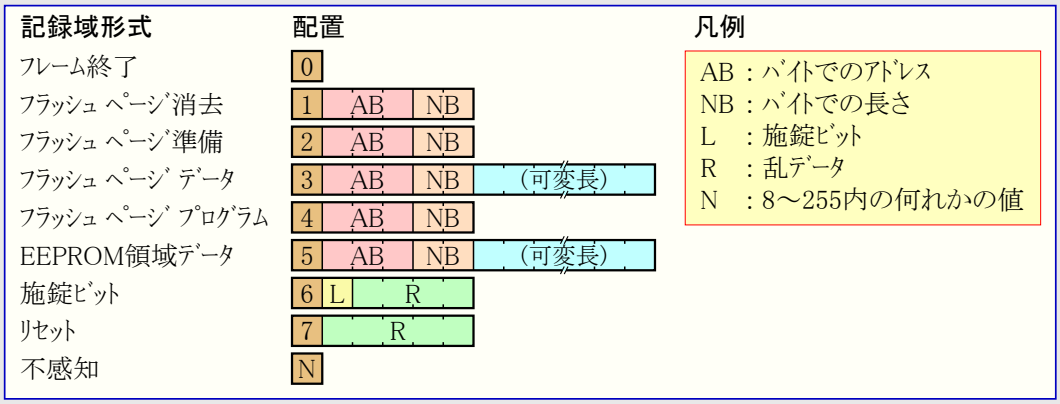
図4-2. CREATE応用の流れ図



4.5.5. 暗号化されたﾌｧｲﾙ

ﾌﾗｯｼｭとEEPROMのﾌｧｲﾙは暗号化されて1つの目的対象ﾌｧｲﾙに格納されます。けれども、暗号化の前にﾃﾞｰﾀは記録域内に編成されます。図4-3.で図解されるように、7つの記録域形式があります。

図4-3. 暗号化ﾌｧｲﾙ用記録域

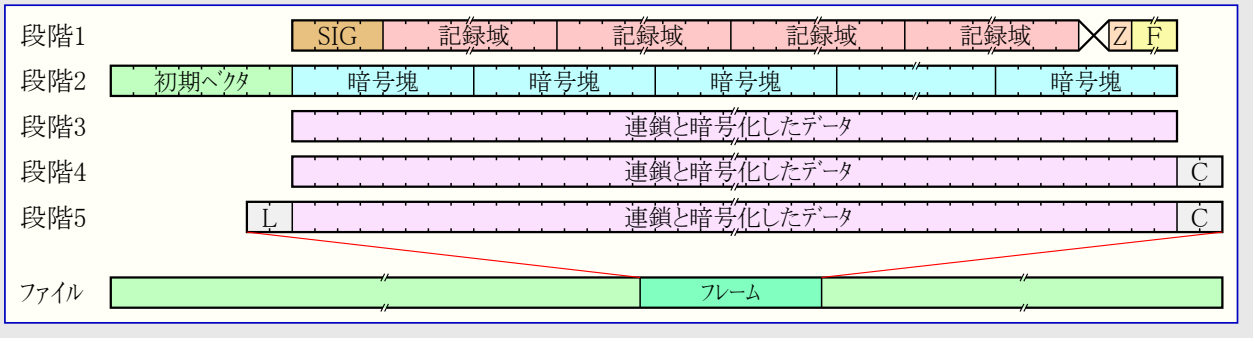


記録域形式はその記録域の先頭ﾊﾞｲﾄとして与えられます。応用ﾃﾞｰﾀは記録域形式1,2,3,4(即ち、消去、準備、ﾌﾗｯｼｭのﾍﾟｰｼﾞ緩衝部への読みと書き)に分類されます。EEPROM領域に関するﾃﾞｰﾀは記録域形式5内に構成されます。施錠ﾋﾞｯﾄは記録域形式6で送られます。記録域形式0と7は各々ﾌﾚｰﾑと転送の終了用です。

他の全ての記録域、即ち7を超える記録域識別子を持つそれらは不感知形式です。この任意選択が許可される(Createﾂｰﾙをご覧ください)と、不感知記録域の乱数はこのﾌｧｲﾙ内の散らされた不定位置に配置されます。

出力ﾌｧｲﾙは図4-4.で図解されるように作成されます。

図4-4. 暗号化ﾌｧｲﾙ作成



この段階は以下で記述されます(番号は図4-4.参照)。

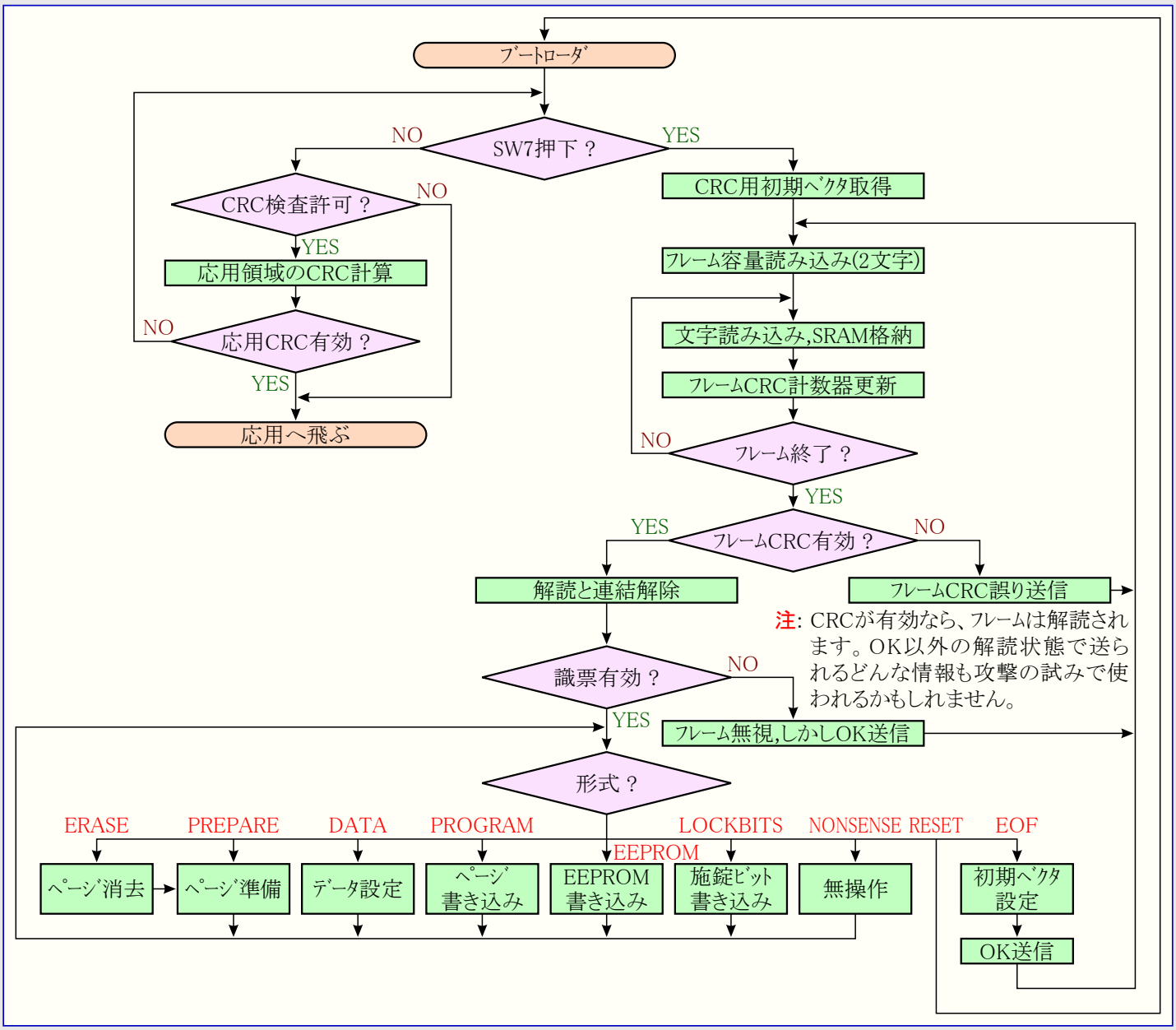
1. ﾃﾞｰﾀは記録域内に構成され、それはその後にﾌﾚｰﾑ識票(SIG)に後続して整列されます。ﾌﾚｰﾑの最後を記すために0(Z)が追加され、ﾌﾚｰﾑは16ﾊﾞｲﾄの倍数のﾌﾚｰﾑ容量を作成するために乱ﾃﾞｰﾀ(F)で穴埋めされます。
2. 初期ﾍﾞｸﾀがﾌﾚｰﾑに添えられます。先頭ﾌﾚｰﾑではそのﾍﾞｸﾀが構成設定ﾌｧｲﾙで与えられるものと等価です。後続ﾌﾚｰﾑでは初期ﾍﾞｸﾀが直前のﾌﾚｰﾑの最後の暗号塊と等価です。
3. 初期ﾍﾞｸﾀと暗号塊は連結されて暗号化されます。初期ﾍﾞｸﾀはその後にﾌﾚｰﾑから取り去られます。
4. CRC-16ﾁｪｯｸｻﾑ(C)が生成されてﾌﾚｰﾑに追加されます。
5. 長さ情報を除いたﾌﾚｰﾑの長さ(L)が計算されて、そのﾌﾚｰﾑの先頭に保存されます。

ﾌﾚｰﾑは出力ﾌｧｲﾙに書かれ、ﾃﾞｰﾀが処理されてしまうまでこの手順が繰り返されます。

4.6. AVRﾌﾞｰﾄﾛｰﾀﾞ

ﾌﾞｰﾄﾛｰﾀﾞはﾃﾞﾊﾞｲｽが暗号化されたﾌｧｰﾑｳｪｱで更新され得る前に目的対象AVRに存在しなければなりません。ﾌﾞｰﾄﾛｰﾀﾞはPCと通信をしてﾌﾗｯｼｭ ﾒﾓﾘの応用領域とEEPROMをﾌﾟﾛｸﾞﾗﾐﾝｸﾞする能力があります。この応用記述と共に含まれるﾌﾞｰﾄﾛｰﾀﾞはIAR™ Embedded Workbench 3.20c版を用いて作成されていますが、他のCｺﾝﾊﾟｲﾗへ移すことができます。ﾌﾟﾛｸﾞﾗﾑの流れは次図で図解されます。

図4-5. AVRﾌﾞｰﾄﾛｰﾀﾞのﾌﾟﾛｸﾞﾗﾑの流れ図



4.6.1. 鍵とﾍｯﾀﾞ ﾌｧｲﾙ

ﾌﾞｰﾄﾛｰﾀﾞをｺﾝﾊﾟｲﾙし得る前に、構成設定が必要ないくつかのﾊﾟﾗﾒｰﾀがあります。始めるにはPC応用のCreateによって生成された暗号鍵とﾍｯﾀﾞ ﾌｧｲﾙがﾌﾞｰﾄﾛｰﾀﾞのﾌｫﾙﾀﾞに複写されなければなりません。このﾌｧｲﾙはﾌﾞｰﾄﾛｰﾀﾞのｿｰｽ ｺｰﾄﾞ内部の#include擬似命令でそれらが参照される時にｲﾝｸﾙｰﾄﾞされます。

4.6.2. ﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙ

この応用記述は以下のﾃﾞﾊﾞｲｽ用のﾃﾞﾊﾞｲｽ特定ﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙと共にやって来ます。

- Atmel ATmega8
- Atmel ATmega16
- Atmel ATmega162
- Atmel ATmega169
- Atmel ATmega32
- Atmel ATmega64
- Atmel ATmega128

対応するAVRで予め定義されたﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙを使ってください。一覧にされていないAVRﾃﾞﾊﾞｲｽについては可能な限り近い目的対象ﾃﾞﾊﾞｲｽに合うﾃﾞﾊﾞｲｽのﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙを使い、4.6.3.と4.6.4.の項で記述されるように変更してください。

4.6.3. リンカ ファイル

ブートローダが上位メモリ領域、即ちブートローダ領域(BLS)に属すため、IARコンパイラは修正されたリンカ ファイルが必要です。リンカ ファイルは.xclの拡張子を持ち、各AVRデバイス個別でIARコンパイラと共に供給されます。この応用記述は修正されたbootldr.xclリンカ ファイルと共に来ます。

リンカ ファイルは"Project"⇒"Options"、"XLINK"分野、"include"タブ、"XCL file name"領域下で定義されます。

注: この応用記述と共に来るデバイス特定プロジェクト ファイルでリンカ ファイルが既に構成設定されています。

4.6.4. その他コンパイラ設定

以下の設定は"Project"⇒"Options"下で得られるダイアログ ウィンドウで定義される必要があります。デバイス特定プロジェクト ファイルで既に全ての設定が定義されていることに注意してください。

表4-4. 必要なコンパイラ設定

分野	タブ	設定内容	例
General	Target	目標AVRに合うように"Processor configuration"を設定。	-cpu=m8, AT90mega8
		"Memory model"をSmallに設定。	
		"Configure system using dialogs (not in .XCL file)"のチェックを外す。	
	Library Configuration	"Enable bit definitions in I/O-include files"をチェック。	
AAVR	Preprocessor	目標AVRに合うようにシンボル"INCLUDE_FILE"を定義。	INCLUDE_FILE="iom8.h"
		目標に合うようにシンボルSPMREGを定義。	SPMREG=SPMCR
		64Kバイトより大きなフラッシュ メモリを持つデバイスに対してシンボル_RAMPZ_を定義。	
		アドレス$40以上に配置されるSPM制御レジスタを持つデバイスに対してシンボル_MEMSPM_を定義。	
XLINK	Output	目標をプログラミングできるように出力ファイル形式を定義。Intel-Extendedに設定。	
	#define	目標ブートローダ領域に合うようにシンボルBOOT_SIZEを16進バイトで定義。	BOOT_SIZE=800
		目標のフラッシュ容量に合うようにシンボルFLASH_SIZEを16進バイトで定義。	FLASH_SIZE=2000
		目標の割り込みベクタ表容量に合うようにシンボルIVT_SIZEを16進バイトで定義。	IVT_SIZE=26
		目標のSRAM量に合うようにシンボルRAM_SIZEを16進バイトで定義。	RAM_SIZE=400
		(I/O領域に後続する)SRAMの開始に合うようにシンボルRAM_BASEを16進バイトで定義。	RAM_BASE=60
		コンパイル処理で報告される総SRAM使用に合うようにシンボルAPP_RAM_USEDを16進バイトで定義。	APP_RAM_USED=30A
	Include	"XCL file name"項目下で、"Override default"をチェック。	
		"XCL file name"項目下で、枠内にファイル名を入力。	$PROJ_DIR$¥bootldr.xcl

下の表4-5.は現在支援されているAVRデバイスに関するコンパイラ任意選択のいくつかを要約します。後で説明されるように、ブートローダ開始アドレスがヒューズ設定に依存することに注意してください(表4-6.をご覧ください)。

表4-5. コンパイラ設定参照基準

項目	ATmega8	ATmega16	ATmega162	ATmega169	ATmega32	ATmega64	ATmega128
リンカ ファイル名	bootldr.xcl						
BOOT_SIZE	800						1000
FLASH_SIZE	2000	4000			8000	10000	20000
IVT_SIZE	26	54	70	5C	58	8C	
RAM_SIZE	400				800	1000	
RAM_BASE	60		100		60	100	
APP_SRAM_USAGE	30A	31E				41E	
SPMREG	SPMCR			SPMCSR	SPMCR	SPMCSR	
RAMPZ							○
_MEMSPM						○	○

注: _RAMPZ_と_MEMSPM_のシンボルはどれとも等しく設定されるべきではありません。必要とされる場合、それらは単に定義済みシンボル一覧に含められるべきです。

4.6.5. ブートローダのインストール

ブートローダをコンパイルし、その後にAtmel AVR Studio®を用いてそれを目的対象に書き込んでください。ブートローダを走行する前に以下のヒューズ ビットが次のように構成設定されなければなりません。

- ブートローダ領域の容量。前の方で記述されるように、この領域容量がBOOT_SIZE設定に合うようにヒューズ ビットを設定してください。通常、BLSは語で与えられますが、BOOT_SIZEパラメータがバイトで与えられることに注意してください。
- ブート リセット ベクタ。ブート リセット ベクタは許可されなければなりません。
- 発振器任意選択。発振器ヒューズ ビットはデバイス依存です。それらは構成設定が必要かもしれません(UARTに影響を及ぼします)。

注: 正しい発振器任意選択設定に特別な注意を払ってください。例え小さな誤調整でも、おそらく通信失敗に終わり得ます。

表4-6.は推奨ヒューズ ビット設定を一覧にします。デバイス依存のヒューズ ビットの詳細な説明についてはデータシートをご覧ください。

表4-6. 推奨ヒューズ ビット

ヒューズ ビット名	ATmega8,8515,8535,16,162,169,32,64	ATmega128
BOOTSZ1,0	0,0	0,1
BOOTRST	0	

注: 0=プログラム、1=非プログラムの意味です。

応用メモリとブートローダの両方を保護するように施錠ビットを設定することが推奨されますが、それはヒューズ ビットが設定されてしまった後でだけです。施錠ビットはAVR Studioを用いて設定することができます。ファームウェアが暗号化される時に命令行の引数としてそれらが定義されていれば、ファームウェア更新中にBLS施錠ビットも設定されます。推奨施錠ビット設定は次のとおりです。

- メモリ施錠ビット: これらは認められていないメモリへのアクセスを防ぐように設定されるべきです。デバイスを消去することなく実装書き込み経由でアクセスすることができないことに注意してください。
- ブートローダ領域用保護形態: SPMとLPMの命令はBLSに対して読み書きを許されるべきではありません。これはブートローダを不正にする応用領域内のファームウェアを保護して解読鍵を安全に保ちます。
- 応用領域用保護形態: 応用領域をアクセスするSPMとLPMの命令に関して全く制限を設定すべきではなく、さもなければブートローダがそれをプログラミングできません。

注: デバイスが正しく施錠されない場合にメモリはISPインターフェース経由でアクセスすることができ、ファームウェアを暗号化する全体的な意味がないことを理解することが重要です。

表4-7.は存在するAVRマイクロ コントローラに対する推奨施錠ビット設定を一覧にします。施錠ビットの詳細な説明についてはデータシートをご覧ください。

表4-7. 推奨施錠ビット

ヒューズ ビット名	ATmega8,8515,8535,16,162,169,32,64,128
BLB12:BLB11	0 0
BLB02:BLB01	1 1
LB2:LB1	0 0

4.7. PC応用 – Update

この応用は目的対象に暗号化ファイルを送るのに使われます。データはPC上のシリアル ポート経由で目的対象ハードウェア上のUSARTへ直接的に送ることができます。プログラムの流れは以下の図4-6.で図解されます。

Update応用はCreate応用で生成されたファイルを読みます。このファイルは暗号化したデータの1つまたはより多くの連結されたフレームから成ります。応用は一度に1フレームのデータを送信し、ブートローダからの応答を待つ間、一時停止します。次のフレームは応答が受信された後にだけ送信され、そうでなければ応用はフレームを再送するか、または通信を閉じるかのどちらかです。

Update応用はコマンド プロンプトから走行します。このコマンド プロンプト引数は表4-8.で一覧にされます。

表4-8. Update応用の命令行引数

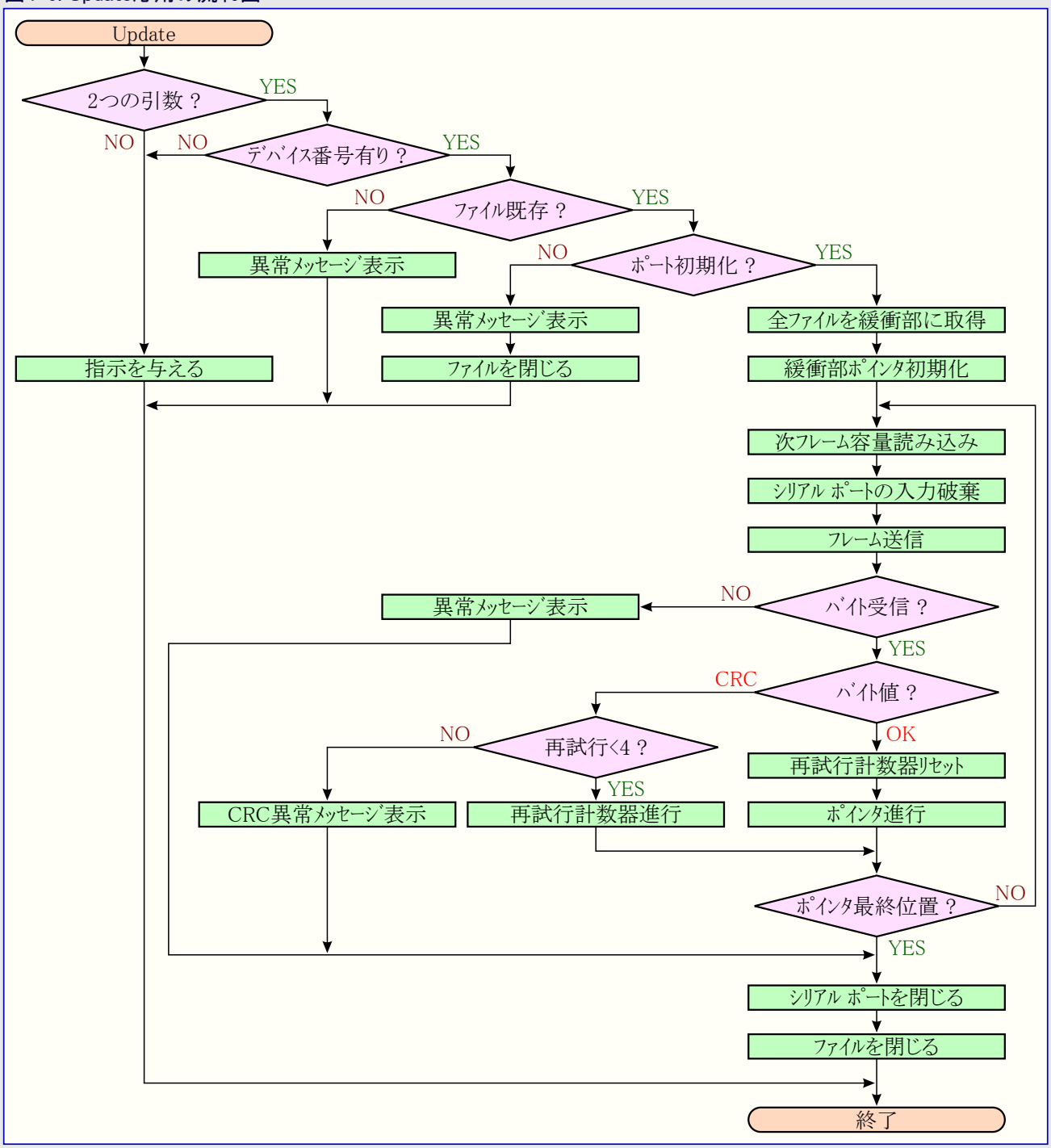
引数	説明
<filename.ext>	転送されるべき暗号化されたファイルへのパス
-COMn	シリアル ポート、nはシリアル ポート番号
-baudrate	ボーレート、ここでのボーレートは実際のボーレート数字

例:

```
update blinky.ext -COM1 -115200
```

更新システムが応用とEEPROMのファイルで指示されたフラッシュ メモリとEEPROMのそれらの部分を更新するだけであることに注意すべきです。応用領域のCRC検査が許可されている、またはcreateツールで消去任意選択が選択されている場合、全応用メモリはプログラミングする(書き込む)前に解除(消去)されます。

図4-6. Update応用の流れ図



4.8. ハードウェア構成設定

目的対象ハードウェアは暗号化したファームウェアがブートローダに送られ得る前に正しく構成設定されなければなりません。この応用記述では目的対象基盤として、外部3.69MHzクリスタル発振器を用いる、Atmel STK®500が使われると仮定されます。STK500は以下のとおりに構成設定されるべきです。

- シリアル ケーブルを用いて("RS232 CTRL"と記されたコネクタ経由で)STK500をPCに接続してください。STK500を電源ONしてください。
- 前で記述されたように、ブートローダをダウンロードして(書き込んで)ヒューズと施錠ビットを設定するのにAtmel AVR Studioを使ってください。STK500を電源OFFしてください。
- シリアル ケーブルを"RS232 SPARE"と記されたコネクタに移動してください。
- デバイスのUSARTのRXDとTXDのピンをRS232 SPAREと記されたコネクタの対応するピンに接続してください。
- PD8(PORTDの8番ピン)をSW7(SWITCHESの8番ピン)に接続してください。
- STK500をONに切り替える間、SW7を押して保持してください。これはブートローダを開始して更新動作形態に設定します。
- SW7スイッチを開放してください。
- 暗号化したデータを目的対象へ送るのに今やPC上のupdate応用を使うことができます。

4.9. 性能

4.9.1.と4.9.2.の項は実行時間とコード量に関してシステム性能を要約します。

4.9.1. 実行時間

受信、復号、データ プログラミング(書き込み)のために目的対象デバイスに必要とされる時間は以下の要素に依存します。

- ファイル容量。より多くのデータは、より長くかかります。
- ボーレート。より高い転送速度は転送時間をより短くします。
- 目的対象AVR速度。より高いクロック周波数は復号時間をより短くします。
- フラッシュ ページのプログラミング(書き込み)時間。これはデバイス不変で変更できません。
- 鍵数。AES128はAES256よりもより速い解読です。実際、AES192はAES256よりも遅いです。それは2のべき乗でない192で行うべき何かがあります。
- その他設定。例えば、応用領域のCRC検査は短い時間がかかります。

4.9.2. コード量

コンパイルに最高最適化設定を用いると、ブートローダは2Kバイトのフラッシュ メモリ内に上手く合います。

暗号鍵が全く与えられない場合、AESなしでブートローダが構築されることに注目すべきです。この応用記述はその後に標準的なブートローダ システムとして実行し、ブートローダ支援を持つどのAVRでも使うことができます。

5. 要約

この応用記述はブートローダ能力を持つAtmel AVRマイクロ コントローラへ安全にデータを転送する方法が提供されます。この資料は保護されたシステムを構築する時に実行されるべき技法も強調されています。AVR設計の安全性を増すために、以下の問題が考慮されるべきです。

暗号化した形式でのダウンロードを支援するブートローダの実装。(製造中に)ブートローダが最初にインストールされる時に将来のファームウェア更新に必要とされる解読鍵が搭載されなければなりません。ファームウェアはその後に部外者から内容を保護し、暗号化形式で配布することができます。

応用とブートローダの領域を保護するためにAVRの施錠ビットを使用。デバイスからの読み込みを防ぐように施錠ビットが設定されると、メモリ内容は取り出すことができません。施錠ビットが設定されない場合、ファームウェアを暗号化する必要がありません。

配布前のファームウェア暗号化。暗号化されたファームウェアは正しい解読鍵なしでどんな外部実体に対しても役に立ちません。

暗号鍵を安全に保護。暗号鍵は2つの場所にだけ格納されるべきで、それは施錠ビットによって保護されているブートローダ内と、製造業者でのファームウェア開発台です。

連鎖暗号化データ。データが連結される時に暗号化された各塊は直前の塊に依存します。結果として、等しい平文は異なる暗号化出力を生成します。

ファームウェアに於いて標準的で予測可能な様式を避ける。殆どのプログラムは侵入者の手助けのために働くだけの低いアドレスへ飛ぶことで始まる割り込みベクタ表のような、共通する枠組みと何らかの予測可能な様式を持ちます。一定の数で未使用領域を穴埋めすることも避けてください。

方法を隠す。使用されつつある算法やどの鍵長かに言及する必要は全くありません。侵入者がシステムについて知らないことが良いことです。暗号化の方法を知ることは或る攻撃者を受け流すと主張されるかもしれませんが、方法について何も知らないことは労力を増して更に多くを防ぐかもしれません。

必要とされるなら、ﾌﾞｰﾄﾛｰﾀﾞは応用領域の消去にも使われるかもしれません。多くの攻撃の試みは通常の動作環境からﾃﾞﾊﾞｲｽを取り外して暴き台での通電を含みます。例えば、LCDが失われている、またはﾒﾓﾘ内にCRC誤りがあることの検出が(ﾌﾞｰﾄﾛｰﾀﾞ領域と解読鍵を含む)全ﾒﾓﾘの完全な消去を始めるかもしれません。

更新用の外部通信ﾁｬﾈﾙを使うことが不適または不可能な応用では、ﾌｧｰﾑｳｪｱをAtmelのCryptoMemory®ﾃﾞﾊﾞｲｽに格納することができます。このﾒﾓﾘは更新が必要とされる時にﾃﾞﾊﾞｲｽのｽﾛｯﾄに簡単に挿入することができるﾘﾑｰﾊﾞﾌﾞﾙ ｽﾏｰﾄ ｶｰﾄﾞとして外装することができます。ﾏｲｸﾛ ｺﾝﾄﾛｰﾗは始動に於いてCryptoMemoryの存在に関して調べ、必要とされる時にﾌｧｰﾑｳｪｱ更新を持ってきます。

安全なﾊｰﾄﾞｳｪｱを使用。ﾊｰﾄﾞｳｪｱが構造的欠陥を持つ場合、強力な暗号規約は役に立ちません。AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗでは安全性の問題が全く報告されていません。

この一覧はもっと長く作ることができますが、その目的は単に設計者を正しい方向に向けさせることです。敵の理解力や忍耐力を見くびらないでください。

6. 参考文献

- Menezes, Oorschot & Vanstone, Handbook of Applied Cryptography
- AES仕様, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
- Atmel Corporation,「AVR230 DES(ﾃﾞｰﾀ暗号化規格)ﾌﾞｰﾄﾛｰﾀﾞ」応用記述
- Brian Gladman, AES Implementation Example, http://gladman.plushost.co.uk/oldsite/cryptography_technology/rijndael/index.php

7. 目次

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
TEL (+1)(408) 441-0311
FAX (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
TEL (+852) 2245-6100
FAX (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parking 4
D-85748 Garching b. Munich
GERMANY
TEL (+49) 89-31970-0
FAX (+49) 89-3194621

Atmel Japan
141-0032 東京都品川区
大崎1-6-4
新大崎勧業ﾋﾞﾙ 16F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ合同会社
TEL (+81)(3)-6417-0300
FAX (+81)(3)-6417-0370

本応用記述はAtmelのAVR231応用記述(doc2589.pdf Rev.2589E-03/12)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。