

AVR230 : DES(データ暗号化規格)ブートローダ

要点

- ブートローダ能力を持つ全てのAVRマイクロコントローラに適合
- ブートローダ能力を持つどのAVRにもコンパイルされたソフトウェアまたは高い慎重性を有するデータの安全な転送が可能
- 簡単な使用と構成設定可能な応用例を包含
 - ・ 2進ファイルとデータの暗号化
 - ・ 目的対象ブートローダ作成
 - ・ 目的対象への暗号化ファイルのダウンロード
- データ暗号化規格(DES)に従った暗号化算法
- 安全性を増すための3重データ暗号化規格(3DES)
- 全AVRデバイスでの2Kバイト内のDESブートローダ
- ATmega128を除く全AVRデバイスでの2Kバイト内の3DESブートローダ
- データの転送、解読、フラッシュメモリ書き込みを含む16Kバイト応用の代表的な更新時間
 - ・ DES、115200bps、16MHz目的対象周波数：20秒
 - ・ 3DES、115200bps、16MHz目的対象周波数：50秒

序説

この応用記述はブートローダ能力を持つ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データ暗号化規格(DES)の使用を含みます。この応用記述は3重データ暗号化規格(3DES)も支援します。

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、ヘッドライヤ、ミシンなどで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、ハードウェアに変更を行うのは難しくなると言ってもよいでしょう。しかしAVRシステム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ブートローダを作ることが可能なように構成設定されます。プログラムメモリ空間はブートローダ領域(BLS)と応用領域の2つの領域に分けられます。両領域はブートローダのコードがBLSで保護され得ると同時に未だ応用領域でコードを更新できるような読み書き保護用の専用施錠ビットを持ちます。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ビ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へ転送されるべきデ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データを暗号化して目的対象AVRにダウンロード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、その複製(それと勿論原型)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。施錠ビットが設定されている場合、その鍵は目的対象AVRから読むことができません。また、この鍵は暗号化されたデータから再生成することもできません。デ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-bit **AVR**[®]
マイクロコントローラ

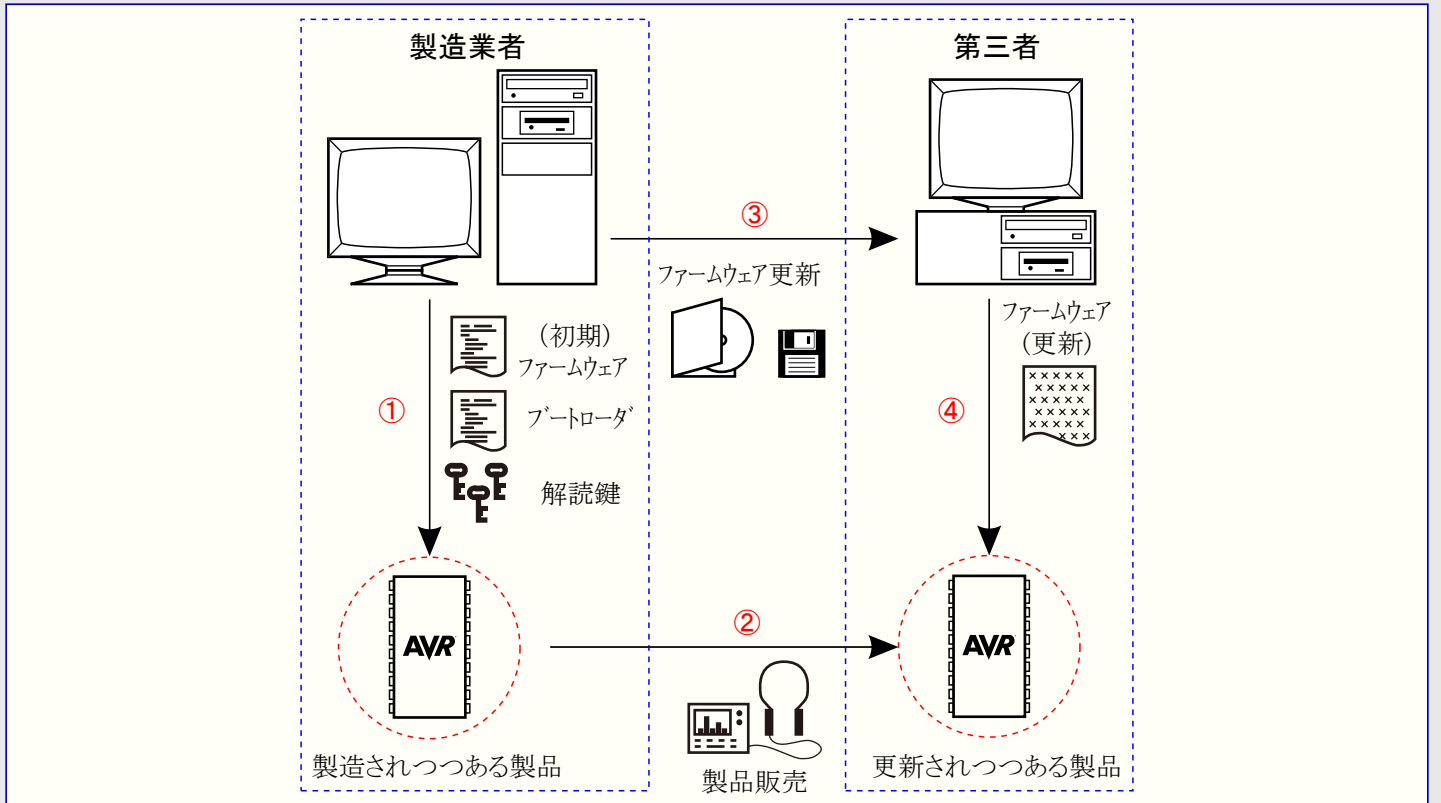
応用記述

本書は一般の方々の便宜のため有志により作成されたもので、Atmel社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 2541D-04/05, 2541DJ3-03/21

図1.は製品がどのように最初に製造され、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. AVRに基づく設計の製造と更新の手順の例



- 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ってそれをフラッシュ 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。ファームウェアは従って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だけ解読されます。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。

理屈

暗号は秘密情報維持の芸術または科学で、暗号法の隠蔽または暗号鍵の保護に基づきます。使う方法の安全性に基づくだけの算法は歴史的興味は主で、実世界の要求には合いません。最新の算法は暗号化と解読の制御に鍵を用います。一致する鍵なしでは掻き混ぜられたメッセージやデータを平文に整列することができません。

暗号化

暗号化はそ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えなくします。

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の選択した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、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。

暗号鍵算法

暗号鍵に基づく算法は対称と非対称の2つの種類に分けられます。対称算法は暗号化と解読に対して同じ鍵を用い、一方非対称算法は違う鍵を用います。最も研究され、多分最も大きく広がった対称算法はDESです。

データ暗号化規格 - DES

データ暗号化規格(DES:Data Encryption Standard)は元来1970代に開発され、後に米国国立標準技術研究所(NIST:US National Institute of Standards)によって標準規格に転化されました。DESは56ビット鍵を用いる対称暗号算法です。この算法は実際問題として非常に強力と証明され、他の多くのものよりも長く残っています。

DES算法は56ビット暗号鍵を使い、これは可能な鍵の組み合わせ数が $2^{56}=72,057,594,037,927,936=7,206 \times 10^{16}$ であることを意味します。

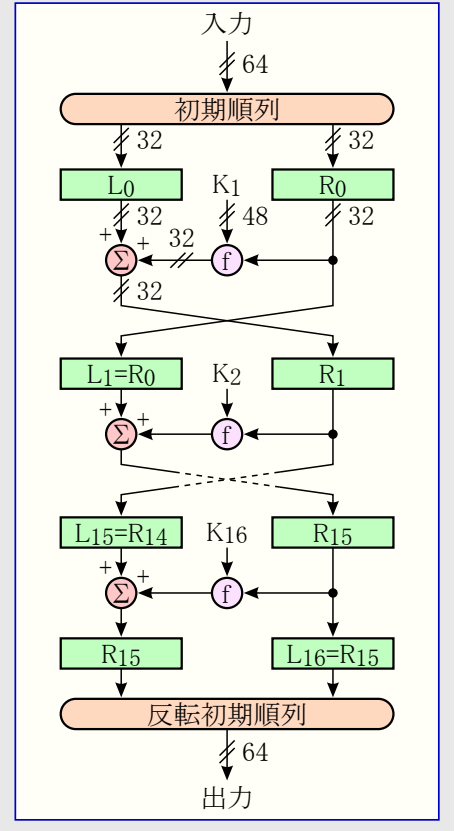
DESは64ビットのデータ塊で操作する塊暗号です。各入力塊は図2.で図解されるように処理されます。

図2.は単一データ塊がどう暗号化されつつあるかを図解します。最初に、入力ビットの順が順列関数に従って変更されます。下位32ビット(R0)はその後に上位32ビット(L0)から分離されて処理されます。暗号鍵の異なる補助組(Kn)を使う各段で、16処理段階があります(図2.では1,2,16段階だけが図解されます)。最後にビット順が初期順列関数に対して逆に変更されます。

解読法は暗号化法と同じで、鍵の補助組(Kn)の順が逆にされるだけです。

DES算法はDES規格でもっと詳細に記述されます(「参考文献」章をご覧ください)。

図2. DES算法に従った暗号化の流れ



3重データ暗号化規格 - 3DES

3重データ暗号化規格(3DES)は3回のDES使用に基き、従って鍵長は56から168ビットに増加します。3DESはDESよりも非常に強力ですが、いくつかのより新しい算法と比べると、実時間応用に関してむしろ遅いです。この応用記述に関してはタイミングが重要でなく、3DESが良いと考えられるかもしれません。

3DES法は3つの56ビット暗号鍵を使います。従って組み合わせ数は $2^{168}=3.741 \times 10^{50}$ に増加します。

3DESはANSI[®] x9.52で定義されるように、3回のDES使用に基づきます。暗号化の流れは右で図解されます。

暗号化中、入力は初めに最初の鍵で暗号化され、その後に第2の鍵で解読され、最後に第3の鍵で暗号化されます。解読中、図4.で図解されるように、鍵の順と符号化/解読塊の順は逆にされます。

規格で以下の鍵操作が定義されています。

図3. 3DES法に従った暗号化の流れ

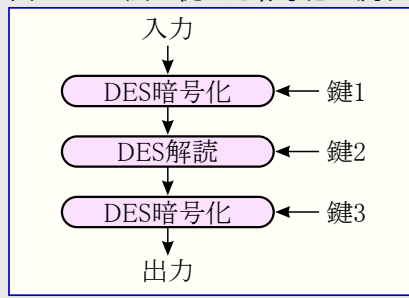


図4. 3DES法に従った解読の流れ

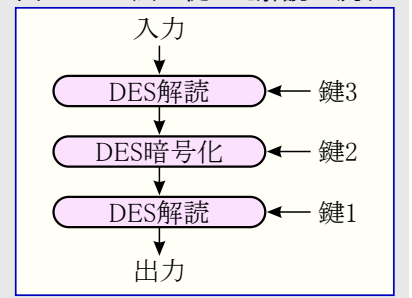


表1. 3DES鍵任意選択

任意選択	独立鍵数	連携する鍵
鍵任意選択1	3 (鍵1、鍵2、鍵3)	(なし)
鍵任意選択2	2 (鍵1、鍵2)	鍵3=鍵1
鍵任意選択3	1 (鍵1)	鍵1=鍵2=鍵3

3DESの完全な暗号長を利用する最初の鍵任意選択1だけが注目されるべきです。3つ目の鍵任意選択3は本質的に単一DESと同じです。

暗号塊連鎖 - CBC

DESと3DES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。64ビットの塊でデータを暗号化するのに56ビット鍵が用いられます。既知の入力塊と一定(とは言え未知)の暗号鍵に関して、その出力塊は常に同じです。この情報は暗号システムの攻撃を欲する誰かにとって有用な提供かもしれません。

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。そのような方法は暗号塊連鎖(CBC: Cipher Block Chaining)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。これは最初に平文の塊と直前の暗号文の塊でXOR操作を実行し、その後に結果を暗号化することによって成し遂げられます。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。

どれくらい安全か?

単に可能な全ての鍵を次々に調べることによって、鍵に基づくどの暗号法も破ることが理論的に可能なことを理解することが重要です。別の方法で破ることができない理想的な系を考慮すると、データ保護のレベルは暗号鍵の長さに指数関数的に比例します。

不正な解読の試みに関する一般的な方法は総当りの使用、換言すると可能な全ての鍵を通して進み、理解できる出力が現れるまで一度に1つずつそれらを試みます。人間の操作者は手動でこのような手順を取れませんが、特別なソフトウェアまたはハードウェアを用いて自動化することができます。これは理解できないものから理解できる出力を区別する方法を作ることができる人間の操作者が未だ必要です。総当たり攻撃では、鍵の長さで指数関数的に増加する鍵を破るのに計算力が必要とされます。平均計算力が時間に渡って指数関数的に増加するため、どの鍵も破られるまでは時間の問題だけです。

表2.は鍵長対暗号強度の結果を説明します。これらの値が指針の意味だけで、予測の多くに於いて大きな変動があることに注意してください。この情報は「参考文献」章で言及される供給元から集められています。

表2. 暗号破り予測時間 対 暗号鍵長 (「参照」をご覧ください。)

鍵長	例	個人、小企業		
		標準的なPC	接続されたPC	特殊なハードウェア
32ビット		数分～数時間		
40ビット		数日～数週間	数分～数時間	
56ビット	DES	実質的に破られない	数週間～数ヶ月	数日～数週間
64ビット			非常に長い期間	短期間
80ビット			実質的に破られない	長期間
128ビット				実質的に破られない
168ビット	3DES			

別の攻撃方法は暗号化された情報がないハードウェアを目標とするものです。この応用記述ではAtmelのフラッシュメモリに基づくAVR構造によってハードウェアが保護されます。解読鍵を含むDES/3DESブートローダはフラッシュメモリ内に格納され、そのメモリはその後施錠ビットを用いて保護されます。ファームウェアは暗号化されたパッチを用いて現場で今や更新することができますが、解読鍵はAVRから取り戻すことができません。

施錠ビットが設定されると、全ての書き込みと外部読み込みアクセスが否定されます。メモリへのアクセスを得る唯一の方法はデバイスを消去することです。それを行うと、施錠ビットが消去される前にブートローダと解読鍵が破壊されます。

システムの安全性改善方法に於ける推奨

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。最初の例では、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。2つ目の例では、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。外来の状況さえ考慮されるべきです。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?。電力が戻って回復され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?。DESに関して総当た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。

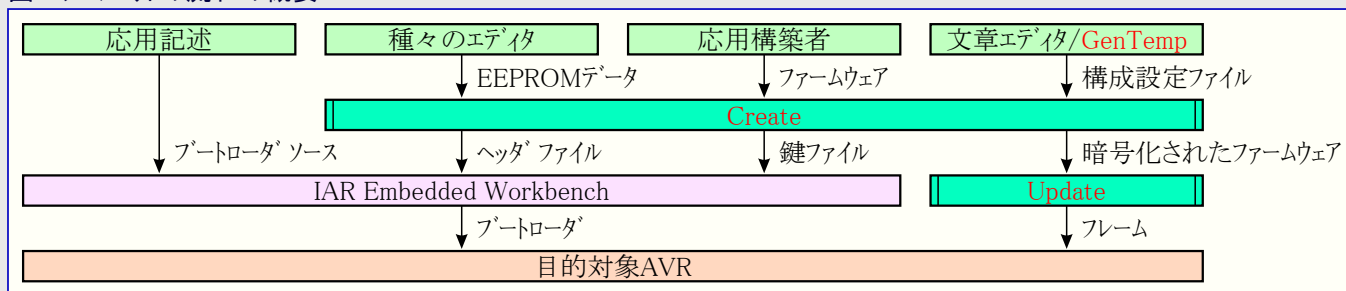
解読処理から得ることができる何れかの還元がありますか?。このような還元も総当たり攻撃基盤構築に於いて攻撃者を手助けし得ます。例えば、ブートローダ内側の解読算法が各塊処理に対して良/否形式の合図を与える場合、この信号は攻撃様式への還元として容易に道筋を定めることができます。

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)。

実装と使い方

本章は応用の構成設定と使用方法を記述します。手順は図5.で図解されます。

図5. プロジェクトの流れの概要



主な段階は次のとおりです。

1. 目的対象AVR用の応用を作成してください。必要ならば、独立したファイルでEEPROM設計(配置)を作成してください。
2. 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。
3. 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。
4. IAR EWを使って、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。
5. 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。
6. 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。

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3.で記述されます。

表3. 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進での最初の暗号鍵。奇数パリティビットとして使われる全ての第8ビットを持つ8つの乱数バイトであるべきです。	なし: 暗号なし	いいえ、しかし強く推奨
KEY2	第2暗号鍵。最初の暗号鍵同様ですが、2鍵3DESに使用。省略されたなら、単一DES暗号が使われます。	なし: 単一DES使用	いいえ、しかし推奨
KEY3	第3暗号鍵。最初の暗号鍵同様ですが、3鍵3DESにだけ使用。省略されたなら、単一または2鍵DES暗号が使われます。	なし: 単一または2鍵DES	いいえ
INITIAL_VECTOR	暗号塊連鎖に使用。16進での8つの乱数バイトであるべきです。	0000000000000000	いいえ、しかし強く推奨
SIGNATURE	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。	00000000	はい
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)。	なし	はい、CRC使用の場合

注: DES算法は56ビット鍵を使いますが、構成設定ファイル内で鍵は8バイト(64ビット)幅で与えられます。これは全ての第8ビットが奇数パリティビットとして使われるためです。

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ATmega16用の試供構成設定ファイルです。

```
PAGE_SIZE      = 128
MEM_SIZE       = 14336
KEY1           = 0123456789ABCDEF
KEY2           = FEDCBA9876543210
KEY3           = 89ABCDEF01234567
INITIAL_VECTOR = 0011223344556677
SIGNATURE      = 89ABCDEF
ENABLE_CRC     = YES
```

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.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. AVR特性要約

特性	ATmega								
	8	8515	8535	16	162	169	32	64	128
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	

PC応用 - GenTemp

この応用記述は構成設定ファイル用の雛形を生成する小さなPC応用を含みます。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)満たされるべきユーザー用の他のパラメータをそのままにします。この手続きが真実の乱数暗号鍵を保証して人の予想を取り去ります。

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どの平文エディタを使っても編集することができます。

PC応用 - Create

このPC応用はMicrosoft®のVisual C++ 6.0版を用いて作成されています。これ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、最初は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回目は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、構成設定ファイルは何時も安全を保たなければならない、最初に使われた後で変更されるべきではありません。

命令行引数

この応用は以下の命令行引数を受け入れます。

表5. 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。

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 DESKeys.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 ファイル名が上で言及したファイル名、換言するとBootLdr.hとDESKeys.incを使うように予めコンパイルされていることに注意してください。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。

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。暗号化に先立って、ソース ファイルはコンパイル、アセンブルされて1つのコード セグメント ファイルと/または1つのEEPROMセグメント 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。

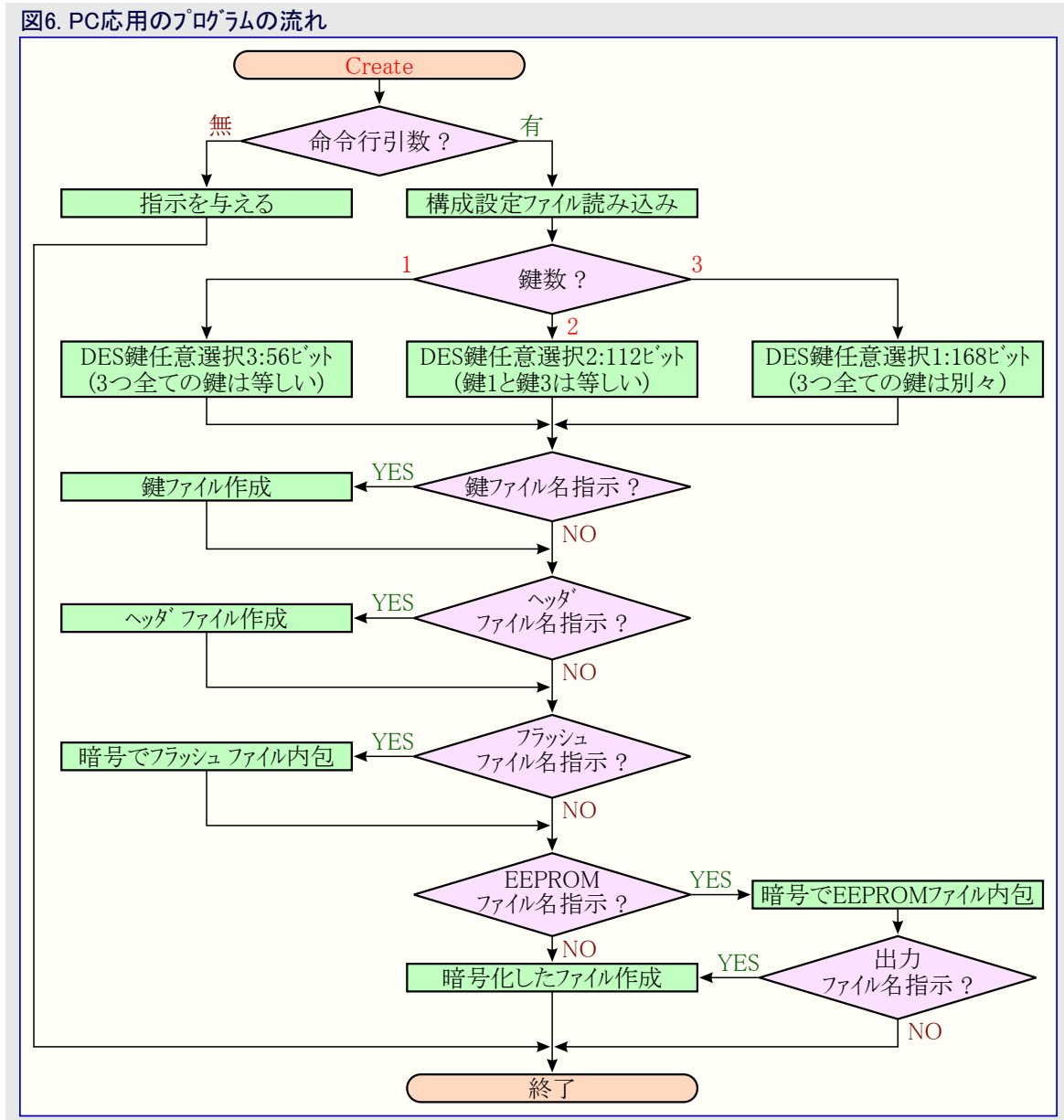
ファイル名はコマンド プロンプトで与えられ、構成設定ファイル内のデータに従って暗号化されたファイルが生成されます。

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。

プログラムの流れ

プログラムの流れが図6.で図解されます。



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、暗号化の前にデータは記録域内に編成されます。図7.で図解されるように、7つの記録域形式があります。

図7. 記録域の形式

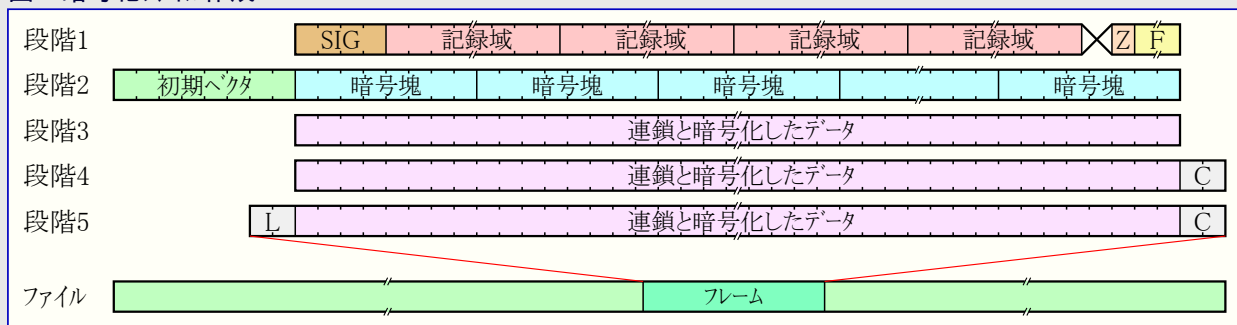
記録域形式	配置	凡例
フレーム終了	0	AB : バイトでのアドレス NB : バイトでの長さ L : 施錠ビット R : 乱データ N : 8~255内の何れかの値
フラッシュ ページ消去	1 AB NB	
フラッシュ ページ準備	2 AB NB	
フラッシュ ページ データ	3 AB NB (可変長)	
フラッシュ ページ プログラム	4 AB NB	
EEPROM領域データ	5 AB NB (可変長)	
施錠ビット	6 L R	
リセット	7 R	
不感知	N	

記録域形式はその記録域の先頭バイトとして与えられます。応用データは記録域形式1,2,3,4(換言すると、消去、準備、フラッシュのページ緩衝部への読みと書き)に分類されます。EEPROM領域に関するデータは記録域形式5内に構成されます。施錠ビットは記録域形式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、換言すると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8.で図解されるように作成されます。

図8. 暗号化ファイル作成



段階は以下で記述されます(番号は図8.参照)。

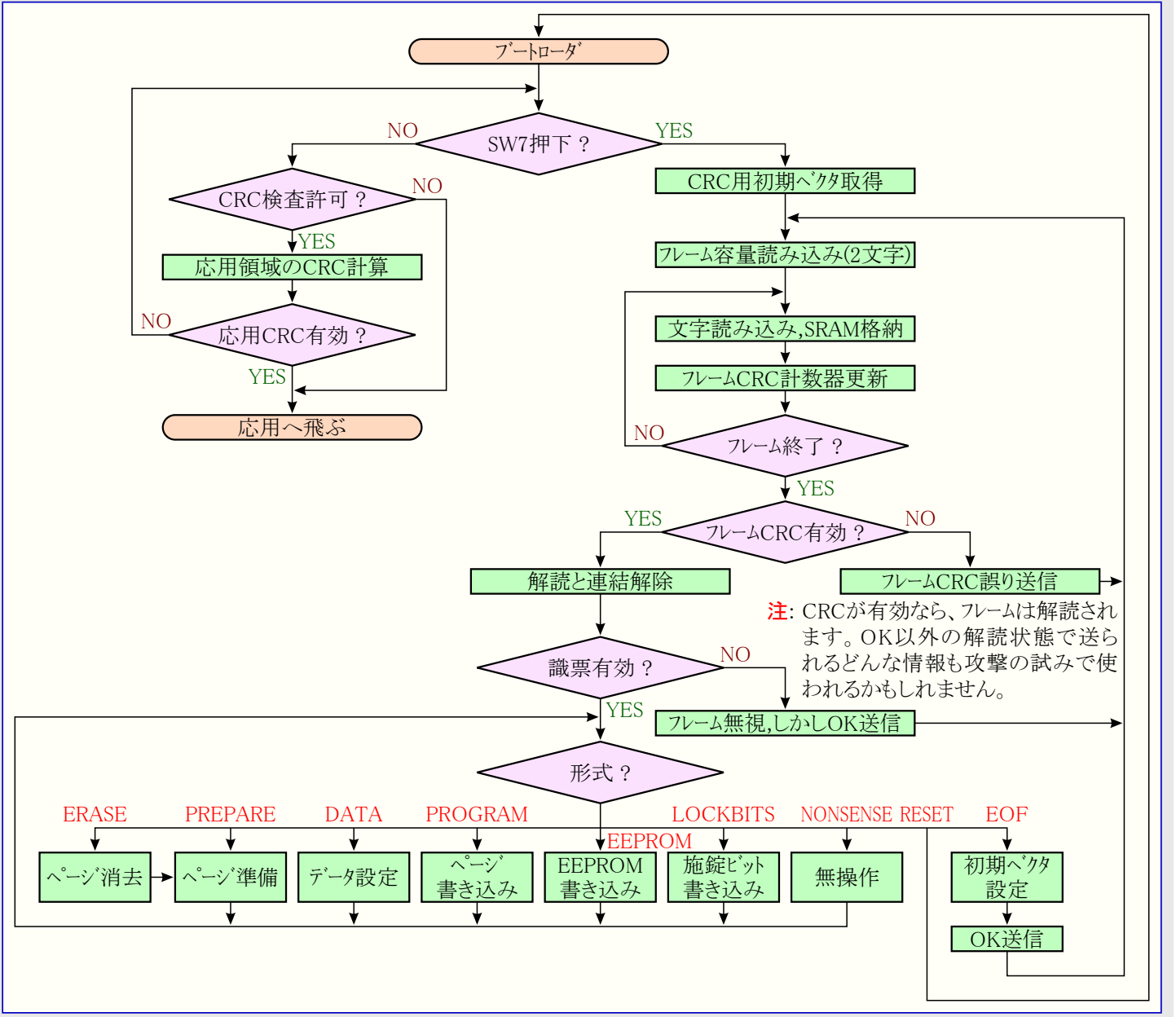
1. データは記録域内に構成され、それはその後にフレーム識別票(SIG)に後続して整列されます。フレームの最後を記すために0(Z)が追加され、フレームは一様な64ビット生成物を作成するために乱データ(F)で穴埋めされます。
2. 初期ベクタがフレームに取り付けられます。先頭フレームではそのベクタが構成設定ファイルで与えられるものと等価です。後続フレームでは初期ベクタが直前のフレームの最後の暗号塊と等価です。
3. 初期ベクタと暗号塊は連結されて暗号化されます。初期ベクタはその後にフレームから取り去られます。
4. CRC-16チェックサム(C)が生成されてフレームに追加されます。
5. 長さ情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、デ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。

ブートローダ

ブートローダはデバ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。ブートローダはPCと通信をしてフラッシュメモリの応用領域とEEPROMをプログラミングする能力があります。この応用記述と共に含まれるブートローダはIAR Embedded Workbench 2.28a版を用いて作成されていますが、他のどんなコンパイラへも移すことができます。プログラムの流れは図9.で図解されま

図9. AVRブートローダのプログラムの流れ



鍵とヘッダ ファイル

ブートローダをコンパイルし得る前に、構成設定が必要ないくつかのパ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ダ ファイルがブートローダにインクルードできるようにブートローダのフォルダに複写されなければなりません。このファイルはブートローダのソースコード内部の#include擬似命令でそれらが参照される時にインクルードされます。

例(これはdes.cソースコードの先頭の複製です。):

```

~
#include "des.h"
#include "bootldr.h"
#include "loader.h"

#if KEY_COUNT > 0

#include "deskeys.inc"
~

```

プロジェクト ファイル

この応用記述は以下のデバイス用のデバイス特定プロジェクト ファイルと共にやって来ます。

- ATmega8
- ATmega8515
- ATmega16
- ATmega169
- ATmega32
- ATmega64
- ATmega128

対応するAVRで予め定義されたプロジェクト ファイルを使ってください。一覧にされていないAVRデバイスについては可能な限り近い目的対象デバイスに合うデバイスのプロジェクト ファイルを使って、下で記述されるように以下の項を変更してください。

リンカ ファイル

ブートローダが上位メモリ領域、換言するとブートローダ領域(BLS)に属するため、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2つのリンカファイルと共に来て、(割り込みベクタ表の選択構成設定に依存して)その1つが使われるべきです。

表6. リンカ ファイル選択

リンカ ファイル名	使用指定
bootldr.xcl	全AVRデバイス、コード用に割り当てられた割り込みベクタ表(メモリ節約)
bootldr_interrupts.xcl	全AVRデバイス、RETIで穴埋めされた割り込みベクタ表

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。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ることに注意してください。

その他コンパイラ設定

ProjectウィンドウでTargetsをReleaseに設定してください。これはデバッグコードなしでブートローダの最小全体コード量を作成します。コード量は2Kバイトのブート領域だけを持つAVRデバイスに対して際どくなります。

注: TargetsがReleaseに設定されていない場合、プロジェクトは必然的に正しくリンクしません。

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7. 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。	
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
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-standardに設定。	
		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
	#define	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
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	

下表は現在支援されているAVRデバイスに関するコンパイル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8.と「表9. 推奨ヒューズビット」をご覧ください)。

表8. コンパイル設定参照基準(推奨ヒューズ設定に於ける)

項目	ATmega8	ATmega8515	ATmega16	ATmega169	ATmega32	ATmega64	ATmega128
リンクファイル名	bootldr.xcl						
BOOT_SIZE	800					800	1000
FLASH_SIZE	2000		4000		8000	10000	20000
IVT_SIZE	26	22	54	5C	58	8C	8C
RAM_SIZE	400	200	400		800	1000	1000
RAM_BASE	60			100	60	100	100

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るでしょう。このソフトウェアは8MHz発振周波数を使うように設計されています。

表9.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9. ヒューズビット (0=プログラム、1=非プログラムの意味です。)

ヒューズビット名	ATmega8	ATmega8515	ATmega16	ATmega169	ATmega32	ATmega64	ATmega128
BOOTSZ1	0						
BOOTSZ0	0					1	
BOOTRST	0						

メモリとブートローダ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るれば、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: これらはメモリへの認められていないアクセスを防ぐように設定されるべきです。
注: メモリが施錠されてしまった後は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ません。
- 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、さもないと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。

下表は存在する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。

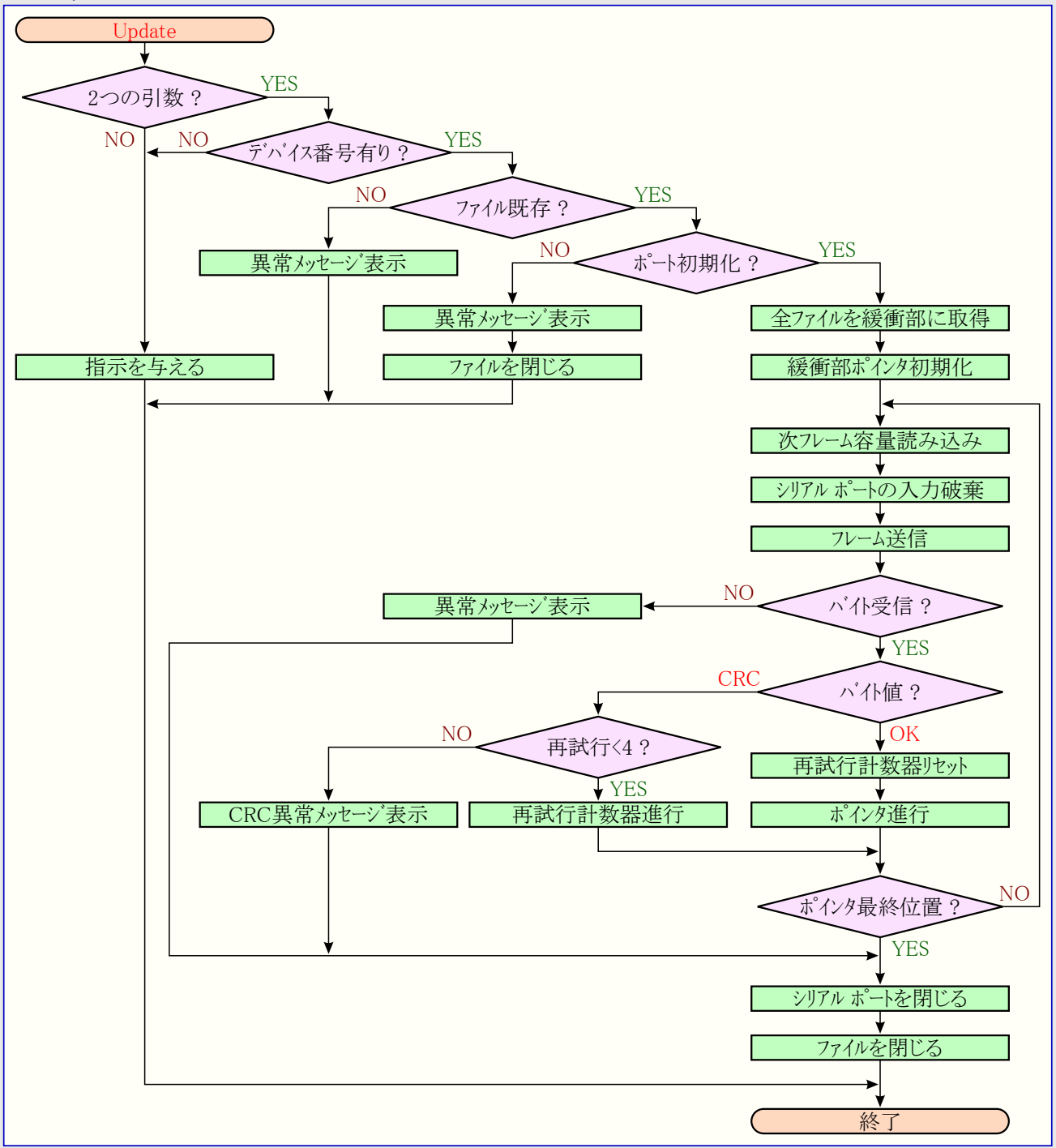
表10. 推奨施錠ビット設定

ヒューズビット名	ATmega8	ATmega8515	ATmega16	ATmega169	ATmega32	ATmega64	ATmega128
BLB12:BLB11	0 0						
BLB02:BLB01	1 1						

PC応用 - Update

Update応用はMicrosoftのVisual C++ 6.0版を使って作成されました。これは目的対象に(EEPROM、フラッシュ、施錠ビットの情報を含む)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図10.で図解されます。

図10. Update応用のプログラムの流れ



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に一時停止します。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11.で一覧にされます。

表11. Update応用のコマンドプロンプト引数

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

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。

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としてSTK500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に繋いでください。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RXD/TXD線をRS232 SPAREと記されたコネクタのRXDとTXDに接続してください。PD0がRXD、PD1がTXDになります。
- 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。

性能

以下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。

速度

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。転送される必要があるより多くのデータは、より長い時間専有します。
- 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。単一鍵DESは3つの鍵の3DESよりも解読がより高速です。
- その他設定。例えば、応用領域のCRC検査は短い時間を専有します。

単一鍵DESについて、全体時間は右式を用いて予測することができます。

ここで FS はバイトでのファイル容量、 BR はボーレート(転送速度)、 f はMHzでのクロック周波数です。これが予測を与えるだけで、例えばATmega128は僅かに異なる(定数60を54によって置換)ことに注意してください。

$$T_{DES} \doteq FS \times \left(\frac{5}{BR} + \frac{1}{60+f} \right) + 3$$

既定設定は9600bpの転送速度と8MHzのクロック周波数です。これは右のように式を簡単化します。

$$T_{DES} \doteq 0.0026 \times FS + 3$$

3DESについて、全体時間は右の式を用いて予測することができます。

$$T_{3DES} \doteq FS \times \left(\frac{5}{BR} + \frac{1}{22+f} \right) + 3$$

既定設定に於いて、式は右のように簡単化することができます。

$$T_{3DES} \doteq 0.0062 \times FS + 3$$

DES解読自体はDESに関して塊当たり245k周期、3DESに関して塊当たり724k周期かかります。これはDESに関して32バイト/s/MHz、3DESに関して10バイト/s/MHzの単位処理量を与えます。

量

下表は目的対象デバイスとコンパイラ任意選択に関してブートローダのコード量を要約します。ATmega128で走行する3DESブートローダを除いて全てのブートローダ任意選択が2Kバイト内に適合することに注目されるかもしれません。

表12. バイトでのブートローダ容量

デバイス	CRC不許可		応用領域のCRC許可		
	DESなし	DES (56ビット)	DES (56ビット)	3DES (112ビット)	3DES (168ビット)
ATmega8	436	1684	1716	1818	1914
ATmega8515	436	1684	1716	1818	1914
ATmega16	476	1752	1786	1888	1984
ATmega169	500	1776	1810	1910	2008
ATmega32	476	1752	1786	1888	1984
ATmega64	504	1810	1844	1946	2042
ATmega128	588	1897	1945	2049	2143

暗号鍵が全く与えられない場合にブートローダがDES/3DES支援なしで構築されることに注目されるべきです。この応用記述はその後標準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。

要約

この応用記述はブートローダ能力を持つ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ています。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、要約として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、暗号化して更新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。使わ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、ブートローダ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デバ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(ブートローダ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[®]デバ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デバイスのスロットに簡単に挿入することができるリムーバブルスマートカード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ブートに於いてCryptoMemoryの存在に関して調べて必要とされる時にファームウェア更新を持てきます。
- 安全なハードウェアだけを使用。ハード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。

この一覧は非常に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。

参考文献

- [1] Douglas Stinson: Cryptography: Theory and Practice, CRC Press, second edition, 1996
- [2] Electronic Frontier Foundation: Cracking DES, 1998:
<http://www.eff.org/descracker.html>
- [3] Electronic Privacy Information Center: <http://www.privacy.org>
- [4] International Association for Cryptologic Research: <http://www.swcp.com/~iacr>
- [5] Man Young Rhee: Cryptography and Secure Data Communications, McGraw-Hill, 1994
- [6] SSH Communications Security: <http://www.ssh.com>



本社

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia

Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe

Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines
Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan

104-0033 東京都中央区
新川1-24-8
東熱新川ビル 9F
アトメル ジャパン株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製造拠点

Memory

2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3
France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR
Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn
Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-76-58-47-50
FAX (33) 4-76-58-47-60

文献請求

www.atmel.com/literature

お断り: 本資料内の情報はAtmel製品と関連して提供されています。本資料またはAtmel製品の販売と関連して承諾される何れの知的所有権も禁反言あるいはその逆によって明示的または暗示的に承諾されるものではありません。Atmelのウェブサイトには位置する販売の条件とAtmelの定義での詳しい説明を除いて、商品性、特定目的に関する適合性、または適法性の暗黙保証に制限せず、Atmelはそれらを含むその製品に関連する暗示的、明示的または法令による如何なる保証も否認し、何ら責任がないと認識します。たとえAtmelがそのような損害賠償の可能性を進言されたとしても、本資料を使用できない、または使用以外で発生する(情報の損失、事業中断、または利益の損失に関する制限なしの損害賠償を含み)直接、間接、必然、偶然、特別、または付随して起こる如何なる損害賠償に対しても決してAtmelに責任がないでしょう。Atmelは本資料の内容の正確さまたは完全性に関して断言または保証を行わず、予告なしでいつでも製品内容と仕様の変更を行う権利を保留します。Atmelはここに含まれた情報を更新することに対してどんな公約も行いません。特に別の方法で提供されなければ、Atmel製品は車載応用に対して適当ではなく、使用されるべきではありません。Atmel製品は延命または生命維持を意図した応用での部品としての使用に対して意図、認定、または保証されません。

© Atmel Corporation 2005. 不許複製 Atmel®、ロコとそれらの組み合わせ、AVR®とその他はAtmel Corporationの登録商標または商標またはその付属物です。他の用語と製品名は一般的に他の商標です。

© HERO 2021.

本応用記述はAtmelのAVR230応用記述(doc2541.pdf Rev.2541D-04/05)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識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。