(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4115759号
(P4115759)

(45)発行日　平成20年7月9日(2008.7.9)　(24)登録日　平成20年4月25日(2008.4.25)

(51)Int.Cl.		F I		
G06F 21/22	(2006.01)	G06F	9/06	660L
G06F 11/10	(2006.01)	G06F	11/10	310B
G06F 21/24	(2006.01)	G06F	12/14	520A
H04L 9/10	(2006.01)	G06F	12/14	530B
		G06F	12/14	540B

請求項の数 9　(全 21 頁)　最終頁に続く

(21)出願番号　特願2002-192422(P2002-192422)
(22)出願日　平成14年7月1日(2002.7.1)
(65)公開番号　特開2004-38394(P2004-38394A)
(43)公開日　平成16年2月5日(2004.2.5)
審査請求日　平成15年9月22日(2003.9.22)

(73)特許権者　000003078
株式会社東芝
東京都港区芝浦一丁目１番１号
(74)代理人　100083806
弁理士　三好　秀和
(74)代理人　100100712
弁理士　岩▲崎▼　幸邦
(74)代理人　100100929
弁理士　川又　澄雄
(74)代理人　100108707
弁理士　中村　友之
(74)代理人　100095500
弁理士　伊藤　正和
(74)代理人　100101247
弁理士　高橋　俊一

最終頁に続く

(54)【発明の名称】耐タンパプロセッサにおける共有ライブラリの使用方法およびそのプログラム

(57)【特許請求の範囲】
【請求項１】
暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも１つの共有ライブラリを呼び出して使用する方法であって、
マイクロプロセッサ上で動作するシステムソフトウェアが、共有ライブラリ用のタスクを作成し、
前記システムソフトウェアが、このタスクにタスク識別子を割り当て、
前記システムソフトウェアが、共有ライブラリのヘッダから、暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵を取得し、
前記システムソフトウェアが、前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、
前記システムソフトウェアが、共有ライブラリ内のローダーを実行し、
共有ライブラリ内のエントリポイントに制御を渡し、呼出元であるシステムソフトウェアに制御を返す
ことからなる共有ライブラリの使用方法。
【請求項２】

請求項１記載の共有ライブラリの使用方法であって、
前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵をマイクロプロセッサ内のテーブルの領域に格納した後で、共有ライブラリ内のインポートテーブルを参照して他の共有ライブラリをロードすることを特徴とする共有ライブラリの使用方法。
【請求項３】
暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも１つの共有ライブラリを呼び出して使用する方法であって、
マイクロプロセッサ上で動作するシステムソフトウェアが、共有ライブラリ用のタスクを作成し、
前記システムソフトウェアが、このタスクにタスク識別子を割り当て、
前記システムソフトウェアが、共有ライブラリのヘッダから、暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵を取得し、
前記システムソフトウェアが、前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、
前記システムソフトウェアが、共有ライブラリ内のローダーを呼出元の個数だけ実行して初期化を行ない、
前記システムソフトウェアが、共有ライブラリが使用するデータを暗号化するためのデータ鍵を呼出元の個数だけ作成し、
前記システムソフトウェアが、前記データ鍵を上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられた前記テーブルの領域に格納し、
共有ライブラリ内のエントリポイントに制御を渡し、呼出元であるシステムソフトウェアに制御を返し、共有ライブラリは待機状態に移行し、
呼出元である前記システムソフトウェアが共有ライブラリと認証をし、
呼出元である前記システムソフトウェアが共有ライブラリが作成した共有メモリ領域のアドレスを受け取り、
前記システムソフトウェアが、上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設定し、
呼出元である前記システムソフトウェアが共有ライブラリから共有ライブラリ内のサブルーチン呼出信号を受け取り、
前記システムソフトウェアが、呼出元のプログラムが渡すデータについてのチェックサムによる整合性の検査を行い、
前記システムソフトウェアが、呼出元とのプログラムが渡すデータについてのチェックサムが共有ライブラリのデータのチェックサムと一致した場合に要求された処理を行ない、
前記システムソフトウェアが、上記処理の結果を前記共有暗号化データ領域にプログラムが渡すデータについてのチェックサムを加えて送付する
ことからなる共有ライブラリの使用方法。
【請求項４】
請求項３記載の共有ライブラリの使用方法であって、
共有ライブラリのサブルーチンが、呼出元のプログラムが渡すデータについてのチェックサムによる整合性の検査を行った後で、呼出元から要求された処理を行うための作業用のメモリ領域を共有ライブラリのデータ鍵で暗号化し、
前記システムソフトウェアが、この暗号化された作業用のメモリ領域内で呼出元から要求された処理を行う
ことを特徴とする共有ライブラリの使用方法。

【請求項５】
請求項３記載の共有ライブラリの使用方法であって、
前記システムソフトウェアが、前記呼出元との認証は、共有ライブラリに予め格納されている秘密鍵による署名を付加した認証情報を呼出元へ送付し、呼出元との間で使用する共通鍵を作成することで行われることを特徴とする共有ライブラリの使用方法。
【請求項６】
請求項５記載の共有ライブラリの使用方法であって、
前記システムソフトウェアが、前記呼出元と認証を行った後で前記共通鍵を共有ライブラリ内の前記データ鍵で暗号化し、
前記システムソフトウェアが、このデータ鍵で暗号化された共通鍵を共有ライブラリ以外から読み出せない秘匿データ領域に格納する
ことを特徴とする共有ライブラリの使用方法。
【請求項７】
暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも１つの共有ライブラリを呼び出して使用する方法であって、
マイクロプロセッサ上で動作するシステムソフトウェアが、共有ライブラリ用のタスクを作成し、
前記システムソフトウェアが、このタスクにタスク識別子を割り当て、
前記システムソフトウェアが、共有ライブラリのヘッダから、暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵を取得し、
前記システムソフトウェアが、前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納し、
前記システムソフトウェアが、共有ライブラリ内のローダーを実行して初期化を行ない、
前記システムソフトウェアが、前記マイクロプロセッサ内の乱数生成部から取得した乱数から当該共有ライブラリの作業用のメモリ領域を暗号化する為のデータ鍵を生成し、
共有ライブラリ内のに制御を渡し、呼出元であるシステムソフトウェアに制御を返し、共有ライブラリは待機状態に移行し、呼出元からの呼出を待機する
ことからなる共有ライブラリの使用方法。
【請求項８】
暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、
マイクロプロセッサ上で動作するシステムソフトウェアが、共有ライブラリ用タスクを作成するステップと、
前記システムソフトウェアが、このタスクにタスク識別子を割り当てるステップと、
前記システムソフトウェアが、共有ライブラリのヘッダから、暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵を取得するステップと、
前記システムソフトウェアが、前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納するステップと、
前記システムソフトウェアが、ローダーを実行して初期化を行うステップと、
エントリポイントに制御を渡し、呼出元であるシステムソフトウェアに制御を返し、共有ライブラリは待機状態に移行するステップ
を実行させるためのプログラム。
【請求項９】

暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、
マイクロプロセッサ上で動作するシステムソフトウェアが、共有ライブラリ用のタスクを作成するステップと、
前記システムソフトウェアが、このタスクにタスク識別子を割り当てるステップと、
前記システムソフトウェアが、共有ライブラリのヘッダから予め記録されている、暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする命令鍵を取得するステップと、
前記システムソフトウェアが、前記暗号化されている共有ライブラリをマイクロプロセッサで復号化可能にする、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納するステップと、
前記システムソフトウェアが、共有ライブラリ内のローダーを実行して初期化を行うステップと、
前記システムソフトウェアが、上記ローダーで初期化をする際にこの共有ライブラリが使用するデータを暗号化するデータ鍵を作成するステップと、
前記システムソフトウェアが、このデータ鍵を上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられた前記テーブルの領域に格納し、
前記システムソフトウェアが、共有ライブラリ内のエントリポイントに制御を渡し、アプリケーションへ制御を返すステップと、
呼出元である前記システムソフトウェアが共有ライブラリと認証を行うステップと、
呼出元である前記システムソフトウェアが共有ライブラリが作成した共有メモリ領域のアドレスを受け取るステップと、
前記システムソフトウェアが、上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設定するステップと、
呼出元である前記システムソフトウェアが共有ライブラリから共有ライブラリ内のサブルーチン呼出信号を受け取るステップと、
前記システムソフトウェアが、呼出元のプログラムが渡すデータについてのチェックサムによる整合性の検査をするステップと、
前記システムソフトウェアが、呼出元とのプログラムが渡すデータについてのチェックサムが共有ライブラリのデータのチェックサムと一致した場合に要求された処理を行うステップと、
前記システムソフトウェアが、上記処理の結果を前記共有暗号化データ領域にプログラムが渡すデータについてのチェックサムを加えて送付さるステップと
を実行させるためのプログラム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、マルチタスクのプログラム実行環境を支援する機能を持つマイクロプロセッサで、プログラムやデータの暗号化及び復号化機能を有し、プログラムの実行コードや処理対象であるデータの秘匿性の保護、及び改竄の防止を可能とするマイクロプロセッサにおける、共有ライブラリの使用方法およびそのプログラムに関する。
【０００２】
【従来の技術】
近年のコンピュータシステムでは、パーソナルコンピュータをはじめとして、様々なメーカーのハードウェアやソフトウェアを組み合わせて構築される、オープンシステムが広く普及している。オープンシステムでは、ハードウェアやシステムプログラム（又はオペレーティングシステム、以下OSと呼ぶ）の情報が開示されており、エンドユーザがこの情報をもとにシステムプログラムを改変・改竄することも原理的に可能である。
【０００３】

このような環境で実行されるアプリケーションプログラムにおいて、その提供者が、プログラムを解析や改竄から完全に保護することは困難である。アプリケーションプログラムはOSの管理下で動作するため、このOS自体が改変されて攻撃の手段として使われた場合、これを避ける手段がない。

【０００４】

そこで予め、オープンシステムで動作するアプリケーションプログラムの解析を防止するために、これを暗号化するという方法がある。プログラムを暗号化すると、解析が困難になるだけでなく、そのプログラムを改変した場合の動作が予測困難となるため、改竄の防止にも有効である。ただし暗号化したアプリケーションプログラムを既存のコンピュータで実行することはできないので、これを復号化しつつ実行するようなマイクロプロセッサが必要となる。このマイクロプロセッサは、OSがアプリケーションプログラムに対して敵対動作をとることを前提として、プログラムの秘密を守るものでなければならない。

【０００５】

このような要件を満たすマイクロプロセッサとしては、橋本他による特開2001-230770や、Lie他による"Architectural Support for Copy and Tamper Resistant Software" （Computer Architecture News 28(5) p. 168-）がある。これらで提案されているマイクロプロセッサは、プログラムだけでなくプログラムが扱う情報やデータも暗号化して、解析や改変から保護する機能を備えている。また、保護を受けたプログラムを、複数同時に疑似並列実行するマルチタスクのプログラム実行環境を提供する。以下、このようなマイクロプロセッサを耐タンパマイクロプロセッサと呼ぶ。

【０００６】

従来提案されてきた耐タンパマイクロプロセッサでは、アプリケーションプログラムは単体で動作し、その実行コードだけで必要な全ての処理が可能であるものとみなされてきた。また、複数のプログラムが協調動作をするためデータ領域の共有方法は寺本他による特願2000-402672で提案されている。しかし、この場合でも協調動作をしている各プログラムは、それぞれ独立した個別のプログラムである。

【０００７】

一方、現代のOSでは共有ライブラリが使われることが多い。ライブラリとは、プログラムを構成する部分であるサブルーチン（プログラムにおいてなんらかの機能を有する一纏まりの命令群）等のサブプログラムを集めたものである。アプリケーションプログラムのプログラマは、アプリケーションプログラムの動作に必要な全ての機能をアプリケーションプログラムに実装するかわりに、一部の機能をライブラリに用意されたサブプログラムに依存することができる。ライブラリとアプリケーションプログラムは、それぞれ別々に開発してから後で自由に組み合わせて使用することができるため、開発を効率化することができる。

【０００８】

古典的なライブラリは、アプリケーションプログラムの作成時にアプリケーションプログラムにリンクされ、ライブラリのサブプログラムはアプリケーションプログラムと一体となって配布される。一方、今日広く使われているライブラリは、アプリケーションプログラムとは別ファイルとして配布される。

【０００９】

共有ライブラリの場合、アプリケーションプログラムとのリンクは、ユーザがプログラムを実行する時に初めて行なわれる。またこのリンク動作は、アプリケーションプログラムの実行可能オブジェクトファイルに対してではなく、アプリケーションプログラムのメモリ上のイメージに対して行なわれる。いったんアプリケーションプログラムと共有ライブラリ間のリンクが行なわれると、共有ライブラリのサブプログラムは通常の共有ライブラリのサブプログラムと同様、アプリケーションプログラム本体から自由に呼び出して使用することができる。

【００１０】

共有ライブラリを使う利点として、必要な記憶領域を節約できることが挙げられる。１つ

のアプリケーションプログラムとそれが使う共有ライブラリのサイズを合計すれば、共有ライブラリを使わない場合に比べてサイズが小さくなることはない。しかし、同じ共有ライブラリを使用するアプリケーションプログラムが複数ある場合は、共有ライブラリのコピーは１つだけあれば良いので、全体として記憶領域の節約になる。この記憶領域の節約は、アプリケーションプログラムの配布ファイルが置かれる二次記憶（ディスク等の外部記憶装置）にも、アプリケーションプログラムが実行時に置かれるコンピュータのメインメモリにも適用できる。

【００１１】

共有ライブラリの中でも動的リンク型と呼ばれる種類のもの（動的リンク共有ライブラリ）は、アプリケーションプログラムを変更することなく共有ライブラリを変更できることを特徴としている。動的リンク共有ライブラリを使うと、ある共有ライブラリを別のものと置き換えることにより、アプリケーションプログラム本体を変更することなくその機能の一部分を変更したり、不具合を修正したりすることが可能である。また、アプリケーションプログラムが実行開始後に使用可能な共有ライブラリを検索し、これをロードするようにすれば、共有ライブラリのみを別途提供することにより、アプリケーションプログラム本体に変更を加えることなく機能追加が可能になる。このような使用形態の共有ライブラリは、しばしばプラグインと呼ばれる。

【００１２】

【発明が解決しようとする課題】

これまで、上述した耐タンパマイクロプロセッサ上で、共有ライブラリを使用できるようなアーキテクチャの提案は行なわれていない。

【００１３】

耐タンパマイクロプロセッサ上で共有ライブラリを実装するためには、以下に述べる要件を満たす必要がある。すなわち、アプリケーションプログラムから共有ライブラリのルーチンを呼び出すことが可能であること。および、ルーチンを呼び出す際にデータをルーチンに渡し、ルーチンから復帰するときも呼出元に処理結果のデータを返すことが出来ること、が要求される。

【００１４】

また上記に加えて、耐タンパマイクロプロセッサが提供するデータ等の保護機能を有効に機能させるためには、アプリケーションプログラムと共有ライブラリとの間でやりとりされる情報の秘密をOS等から守る必要がある。ルーチン呼出の際に秘匿したいデータをやりとりする場合、データを渡す前に相手が信用できるかどうか（OS等によって悪意のある別の共有ライブラリに置き換えられていないかどうか）を認証する必要がある。また、この認証が終った後で呼出先のルーチンがすり替えられることも防止しなければならない。共有ライブラリが複数のアプリケーションプログラムから同時に使用される場合は、共有ライブラリが１つのプログラムの秘密を他のプログラムに漏らすようなことはあってはならない。

【００１５】

また共有ライブラリは、任意のアプリケーションから使用可能でなければならない。つまり認証により、ある特定のアプリケーションプログラムからしか使用できないような共有ライブラリは、用途が限定されるため共有ライブラリとしては不十分である。一方、共有ライブラリを使用するアプリケーションプログラムの側は、共有ライブラリに秘匿にしたいデータを渡す前に、共有ライブラリがこのデータを他に漏らさないことを確認できることが望ましい。

【００１６】

このためアプリケーションプログラムが、共有ライブラリを認証する手段が用意されていなければならない。

【００１７】

共有ライブラリの動作は、アプリケーションプログラムからデータを受け取り、これに対しある処理をして、その結果をアプリケーションプログラムに返すことである。このとき

、アプリケーションプログラムから受け取ったデータや、その処理結果を、アプリケーションプログラムと共有ライブラリ以外の第三者に漏洩してはならない。すなわち、データの受渡しを暗号化して行なわなければならないだけでなく、受け取ったデータに対応する処理結果を返す先のプログラムが、元のデータを提供したアプリケーションプログラムと同一であることを確かめる必要もある。

【００１８】

また、誰でもこの共有ライブラリを使うアプリケーションプログラムを書くことができるため、悪意のあるアプリケーションプログラムから使用される可能性がある。その場合でも共有ライブラリの内部動作を解析から防御できなければならない。つまり、共有ライブラリの実行コードをアプリケーションプログラムが読み出したり、共有ライブラリに渡されたデータが処理される途中過程のデータをアプリケーションプログラムが盗み見たりすることを、防止できなければならない。

【００１９】

本発明は従来の耐タンパマイクロプロセッサでは使用することのできなかった共有ライブラリを使用可能にすること。および、上記要件を満たしつつ、かつ前述した利点の得られる共有ライブラリの使用方法、およびそのプログラムを提供することを目的とする。

【００２０】

【課題を解決するための手段】

上記目的を解決する、本発明の第一の側面は以下を要旨とする。暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも１つの共有ライブラリを呼び出して使用する方法であって、まず、共有ライブラリ用のタスクを作成し、次にこのタスクにタスク識別子を割り当てる。次に共有ライブラリのヘッダから予め記録されている命令鍵を取得し、次にこの命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納する。次に共有ライブラリ内のローダーを実行して初期化を行ない、共有ライブラリ内のエントリポイントを経由してアプリケーションへ制御を返す。

【００２１】

このようにして実現された共有ライブラリの使用方法にあっては、共有ライブラリそのものを一つの識別子を有するタスクとして処理し、かつこの識別子に対応するプロセッサ内の場所に共有ライブラリを暗号・復号化する為の命令鍵を記録するので、共有ライブラリの実行コードが保護される。

【００２２】

また本発明における第二の側面は、暗号化されたコードを復号化して実行する機能を有すると共に、少なくとも１つのプログラム及びそのプログラムが呼び出す少なくとも１つの共有ライブラリに対応した複数の暗号鍵を格納する複数の領域からなるテーブルを有するマイクロプロセッサ上で、呼出元プログラムから前記少なくとも１つの共有ライブラリを呼び出して使用する方法であって、以下を要旨とする。まず共有ライブラリ用のタスクを作成し、このタスクにタスク識別子を割り当てる。次に、共有ライブラリのヘッダから、予め記録されている命令鍵を取得し、この命令鍵を前記マイクロプロセッサ内の前記共有ライブラリに割り当てられた識別子に対応するテーブルの領域に格納する。共有ライブラリ内のローダーを実行して初期化を行ない、上記ローダーで初期化をする際にこの共有ライブラリが使用するデータを暗号化するためのデータ鍵を作成する。次いでこのデータ鍵を、上記プロセッサ内の当該共有ライブラリに対応する識別子が割り当てられたの前記テーブルの領域に格納し、共有ライブラリ内のエントリポイントを経由してアプリケーションへ制御を返し、呼出元からの呼出を待機する。呼出元からの呼出後で呼出元と認証を行う。呼出元側でこの認証情報を使用して認証を行ない、認証が正しく行なわれた後呼出元との間で使用する共通鍵を作成する。呼出元が作成した共有メモリ領域のアドレスを受け取り、上記共有メモリ領域を呼出元とのデータ交換に使用する共有暗号化データ領域に設

定する。そして、呼出元から共有ライブラリ内のサブルーチン呼出信号を受け取り、呼出元のチェックサムを検証する。呼出元とのチェックサムが共有ライブラリのチェックサムと一致した場合に要求された処理を行ない、上記処理の結果を前記共有暗号化データ領域にチェックサムを加えて送付する。

【００２３】

このような共有ライブラリでは、上記第一の側面に加えてプログラムから要求された処理を共有ライブラリが行う際に、共有ライブラリのデータ鍵を使用して処理内容および処理結果のデータを暗号化する。このため、共有ライブラリ自身のコードの暗号化に加えて、内部処理方法や処理結果が外部へ漏洩することを防げる。また、処理の要求を受ける時と処理結果を呼出元に返す時に互いのチェックサムが一致するかどうかを検証するため、処理の前後で呼出元がＯＳなどによって変更された場合にも、共有ライブラリ内の処理内容や処理結果を確実に保護できる。また加えて、呼出元の個数だけロードを行うため、一つの共有ライブラリを複数の呼出元から使用することも可能である。

【００２４】

【発明の実施の形態】

以下、図面を参照しながら本発明の一実施の形態を説明する。

【００２５】

図１は、本発明が適用されるマイクロプロセッサ（耐タンパマイクロプロセッサ）１の基本ハードウェア構成図である。マイクロプロセッサ１は、プロセッサ・コア１０と、コード・データ暗号／復号処理部１１と、鍵値テーブル１３と、タスクID格納レジスタ１４と、乱数生成部１５と、外部バスインタフェース１２とを有する。

【００２６】

耐タンパマイクロプロセッサとして使用されるマイクロプロセッサ１が、通常のマイクロプロセッサと大きく異なる点は、その内部にコード・データ暗号／復号処理部１１を有している点である。バス上の実行コードやデータは、暗号化された状態で流れる。プロセッサ・コア１０に入力される実行コードやデータは、コード・データ暗号／復号処理部１１によって復号化されて入力される。またデータがプロセッサ・コア１０から再度バス上へ流れて出るときには、コード・データ暗号／復号処理部１１を経由して暗号化されて出力される。コード・データ暗号／復号処理部１１は暗号化や復号化に暗号鍵を使用する。この暗号鍵は、同プロセッサ１内の鍵値テーブル１３から取得する。また、鍵値テーブル１３の中から使うべき暗号鍵を選択・取得するために、タスクID格納レジスタ１４を使う。

【００２７】

図２は、鍵値テーブル１３の構成を示す図である。鍵値テーブル１３は、暗号鍵の値を格納するレジスタからなる鍵値テーブルエントリ１３０を複数個並べたものである。この鍵値テーブルエントリは、後述する０～ｎ番までのｎ＋１個からなるタスクＩＤの数だけ設けることが可能である。各鍵値テーブルエントリ１３０は、タスクＩＤごとに130_0、130_1、130_2、・・・、130_nのように互いに識別される。鍵値テーブルエントリ$130_{i\ (0 \leqq i \leqq n)}$には、後述するプログラム３の実行コード３２を暗号・復号化するための鍵である命令鍵ｋ１、プログラム３が操作するデータを暗号化・復号化するためのデータ鍵ｋ２、命令鍵ｋ１やデータ鍵ｋ２が適用されるメモリ範囲を示すアドレス情報１３ａを格納することができる。鍵値テーブル１３には複数の鍵値テーブルエントリ130_0、130_1、130_2、・・・、130_nが並んでいるため、それぞれの鍵値テーブル$130_{i\ (0 \leqq i \leqq n)}$に別々のプログラム３の命令鍵を格納することができる。これにより異なる暗号鍵で暗号化された複数のプログラムのインスタンス（プロセス等）を疑似並列動作させるマルチタスク動作に対応可能となる。

【００２８】

タスクID格納レジスタ１４は、タスクIDを１つ格納するレジスタである。単一スレッドのプロセスや、複数スレッドを使うプロセスの中の各スレッド、又は呼び出された共有ライブラリのインスタンス（以下これらのプロセス、スレッドやインスタンスを、タスクと呼ぶ）をマルチタスク動作で疑似並列動作させているとき、現在マイクロプロセッサが実行

中のタスクを識別するために、タスクID格納レジスタ１４を用いる。

【００２９】

乱数生成部１５は、プロセッサ・コア１０からの読み出しに対して、毎回異なる乱数を提供する。この乱数生成部１５はプログラムが暗号化を行うための鍵や、認証のために必要な乱数を生成するために使用することができる。

【００３０】

本発明が適用されるマイクロプロセッサ上で動作するソフトウェアは、システムソフトウェア（OS）と、アプリケーションプログラム３（以下、単にプログラムと呼ぶ）と、共有ライブラリ４からなる。

【００３１】

以下では、アプリケーション３と、共有ライブラリ４の構成等について説明する。図３は、本発明に係るプログラム３の構成図である。プログラム３はヘッダ３１と、実行コード３２と、初期化済みデータ３３と、インポートテーブル３４からなる。ヘッダ３１には、プログラム３の実行コード３２を復号化するための命令鍵ｋ３１が含まれる。インポートテーブル３４は、プログラム３が使用する共有ライブラリ４や、それに含まれるシンボル（シンボルは後述する共有ライブラリ内の各サブルーチン等を識別するための識別子である）を指定するもので、共有ライブラリ４のロードをするために必要な情報である。

【００３２】

図４は、本発明に係る共有ライブラリ４の構成図である。共有ライブラリ４はプログラム３と同様、ヘッダ４１と、実行コード４２と、初期化済みデータ４３と、インポートテーブル４４からなる。共有ライブラリ４のインポートテーブル４４は、共有ライブラリ４自身が別の共有ライブラリを使用する場合に必要な情報を含む。共有ライブラリ４の実行コード４２はさらに、ブートストラップルーチン４２ａと、エントリポイントコード４２ｂと、それぞれ機能を持ったいくつかのサブルーチン４２ｃとからなる。図４では、複数のサブルーチンを添字i(1≦i≦n)で、$42c_1$、$42c_2$、・・・、$42c_n$のように区別してある。

【００３３】

ブートストラップルーチン（ローダー）４２ａは、共有ライブラリ４が呼出元のプログラム３等によってロードされる時に必要な処理を行うためのものである。エントリポイントコード４２ｂは、呼出元のプログラム３等から共有ライブラリ４が使用されるときの呼出先となる共有ライブラリ４のプログラム３に対する入口となる部分である。共有ライブラリ４のその他のサブルーチン$42c_{i(1\leqq i\leqq n)}$が、実際にプログラム３が使用する機能を実装したコードである。共有ライブラリ４の各サブルーチン$42c_{i(1\leqq i\leqq n)}$は、それぞれを共有ライブラリ４の中で識別できるような識別子が付与されている。

【００３４】

また共有ライブラリの実行コード４２内には、共有ライブラリ４が、後述するプログラム３等からの呼出に応じて処理を行う際に、要求された処理を行う際に使用するデータを暗号化するためのデータ鍵ｋ４２も含んでいる。

【００３５】

（タスクの開始）

次に図５を参照しながら、OSがユーザからのプログラム実行の要求や、既存プロセスによる新規プロセスや新規スレッドスレッド作成の要求により、タスクの実行を開始する手順を説明する。

【００３６】

OSは最初にタスクの作成を行う（ステップ５１）。これはメモリの確保や、タスクを管理するためのデータ構造の作成等を含む。また、OSは新しいタスクのためのタスクIDの割当も行う。

【００３７】

次にOSはプログラム３のヘッダ３１より命令鍵ｋ３１を取得し、先ほど割り当てたタスクIDに対応する鍵値テーブルのエントリ$130_{i(0\leqq i\leqq n)}$に命令鍵ｋ３１を格納する（ステ

ップ５２）。

【００３８】

また、OSはプログラム３のインポートテーブル３４を参照し、ここに記述された各共有ライブラリ４を、後述する手段（図６参照）によりロードする（ステップ５３）。インポートテーブル３４には任意の数の共有ライブラリ４が記述されている可能性がある。従って、OSは記述された共有ライブラリ４それぞれに関して、別個にロードの作業を行う。なお、タスク開始前に共有ライブラリ４をロードするかわりに、タスクが開始された後でタスクからの要求に応じて共有ライブラリ４をロードしても良い。

【００３９】

OSは要求された全ての共有ライブラリ４のロードが完了すると、新しいタスクにコンテキスト（ただし、ここでコンテキストとは、プロセスの処理状況や環境が記載されているデータを指す）の切り替えを行ない、タスクの実行を開始する（ステップ５４）。このタスク切り替えの際にプログラムのタスクIDがタスクID格納レジスタ１４に格納される。

【００４０】

耐タンパマイクロプロセッサ１において、プログラムの実行コード３２は次のように実行される。外部バスインタフェース１２が外部メモリより命令コードを取得すると、コード・データ暗号／復号処理部１１は、タスクID格納レジスタ１４に格納されたタスクIDに対応する鍵値テーブル１３の内容を参照し、ここに格納されたプログラム３の命令鍵ｋ３１によって実行コード３２を復号化する。次に復号化された実行コード３２はプロセッサ・コア１０に渡されて実行される。

【００４１】

（共有ライブラリのロード）

図６は、OSが共有ライブラリをロードする手順を示す図である。

【００４２】

通常のマイクロプロセッサでは、共有ライブラリのために呼出元のプログラムとは別のタスクを作成するようなことはしない。しかし耐タンパプロセッサ１では、プログラム３又は共有ライブラリ４の一方に悪意があった場合の安全性を考慮し、共有ライブラリ４にも別のタスクを作成する。このため、OSが共有ライブラリ４をロードするときに最初に行う作業は、タスクの作成及び新しいタスクのためのタスクIDの割り当てである（ステップ６１）。

【００４３】

次にOSは共有ライブラリ４のヘッダ４１より命令鍵ｋ４１を取得し、割り当てられたタスクIDに対応する鍵値テーブル１３のエントリ$１３０_{i(0 \leqq i \leqq n)}$にこの命令鍵ｋ４１を格納する（ステップ６２）。このステップにより予め命令鍵ｋ４１で暗号化されている共有ライブラリ４をマイクロプロセッサ１で復号化可能となるとともに、各命令鍵ｋ４１が別々のエントリ$１３０_{i(0 \leqq i \leqq n)}$に格納される。従って、複数の共有ライブラリ４が存在する場合でも、これらを識別可能となり安全である。

【００４４】

また、OSは必要に応じて共有ライブラリ４のインポートテーブル４４を参照し、ここに記述された他の共有ライブラリ４をロードする（ステップ６３）。このロードの手順は、プログラム３のインポートテーブル３４に従って共有ライブラリ４をロードする場合の手順と同じである。

【００４５】

次に、OSは共有ライブラリ４のブートストラップルーチン（ローダー）４２ａを実行する（ステップ６４）。ブートストラップルーチン４２ａは、必要な初期化の処理を行なった後、エントリポイントコード４２ｂに制御を渡す（ステップ６５）。次いで、エントリポイントコード４２ｂは、制御をOSに戻す。制御がOSに戻ると共有ライブラリ４は待機状態に移行する（ステップ６６）。

【００４６】

なお、ロードしようとしている共有ライブラリ４を、既に実行中の他のタスクが使用して

いる場合であっても、共有ライブラリ４のロードは呼出元のタスク毎に毎回行う。結果として、呼出元タスクと同じ数だけのタスクIDが同じ共有ライブラリ４に対し割り当てられる。ただし共有ライブラリ４の実行コード４２は、OSやマイクロプロセッサ１の仮想メモリの仕組みを利用すれば、呼出元タスクが複数ある場合でも１回だけメモリにロードするだけで良い。

【００４７】

（共有ライブラリ内のサブルーチンの呼出）

タスク実行の途中で、共有ライブラリ４のサブルーチン４２$c_{i(1 \leqq i \leqq n)}$を呼び出す必要があるときは、図７に示すシーケンスによってサブルーチン４２$c_{i(1 \leqq i \leqq n)}$呼出の処理が行なわれる。

【００４８】

図８は、サブルーチン呼出パラメータブロックの構造８である。呼出元のタスクは、呼出先の共有ライブラリ４の識別子８１と、呼び出すサブルーチン４２$c_{i(1 \leqq i \leqq n)}$の識別子８２と、サブルーチン４２$c_{i(1 \leqq i \leqq n)}$に渡すパラメータ８３とを、サブルーチン呼出パラメータブロック８に格納する。サブルーチン呼出パラメータブロック８を作成した後、呼出元のタスクはシステムコールを使い、サブルーチン呼出パラメータブロック８を渡して、OSにサブルーチン呼出の要求を出す。

【００４９】

OSはこの要求を受けると呼出元のタスクを停止させ、サブルーチン呼出パラメータブロック８の共有ライブラリ識別子８１を参照して呼出先の共有ライブラリ４のタスクを選択し、その共有ライブラリ４にタスクの切り替えを行う。このときOSはサブルーチン呼出パラメータブロック８を共有ライブラリのタスクに渡す。共有ライブラリ４にタスクの切り替えを行うと共有ライブラリ４のタスクIDがタスクID格納レジスタ１４に格納される。するとこれまで待機状態であった共有ライブラリ４のエントリポイントコード４２ｂの実行が再開される。

【００５０】

共有ライブラリ４の暗号化された実行コード４２を実行する手順は、上記で説明したプログラム実行コード３２の実行手順と同じである。

【００５１】

共有ライブラリ４のエントリポイントコード４２ｂは、実行が再開されると渡されたサブルーチン呼出パラメータブロック８の内容を参照し、ここに指定されたサブルーチン識別子８１に対応するサブルーチン４２$c_{i(1 \leqq i \leqq n)}$を呼び出す。呼び出されたサブルーチン４２$c_{i(1 \leqq i \leqq n)}$はサブルーチン呼出パラメータブロック８にあるパラメータ８３等を参照し、要求された処理を行う。処理の結果として返すべきデータはサブルーチン呼出パラメータブロック８に格納し、処理が完了するとエントリポイントコードに復帰し、次にここからOSに復帰する。OSに処理が復帰すると共有ライブラリ４は再び待機状態に戻り、OSはサブルーチン呼出パラメータブロック８を呼出元のタスクに返して呼出元タスクの実行を再開する。

【００５２】

（マルチスレッド動作）

共有ライブラリ４の呼出元であるのプログラム３がマルチスレッド動作をする場合、複数のスレッドが同時に同じ共有ライブラリ４のタスクを使用することはできない。

【００５３】

そこで、OSは呼出元プロセスが実行中に新しいスレッドの作成を要求すると、呼出元プログラム３が使う全ての共有ライブラリ４に関して、もう１度ロードの処理を行う。結果として、各共有ライブラリ４には、それぞれ呼出元プロセスのスレッドの数と同じだけの数のタスクIDが割り当てられる。OSはあるスレッドから共有ライブラリ４のサブルーチン４２$c_{i(1 \leqq i \leqq n)}$呼出を要求されると、その共有ライブラリ４に割り当てられた未使用のタスクIDを選択して、このタスクIDを使用して共有ライブラリ４のサブルーチン４２$c_{i(1 \leqq i \leqq n)}$を呼び出す。

【００５４】
また、OSは使用するタスクIDの数を節約するため、共有ライブラリ４のロードをタスクの作成時ではなく、サブルーチン４２ｃ$_{i(1 \leqq i \leqq n)}$呼出のときにタスクIDが不足した場合に行うようにしても良い。
【００５５】
（共有ライブラリに固有の秘匿データの保持方法）
共有ライブラリ４のサブルーチン４２ｃ$_{i(1 \leqq i \leqq n)}$が、共有ライブラリ４内部での処理経過や処理方法等の秘匿データを保持するには次のようにする。
【００５６】
共有ライブラリ４の作成者は、秘匿データ暗号化のための暗号化鍵であるデータ鍵ｋ４２を１つ作り、実行コード４２の中に埋め込む（図４参照）。このデータ鍵ｋ４２の値は実行コード４２の中に埋め込まれており、実行コード４２は命令鍵ｋ４１によって暗号化されている。従って、共有ライブラリ４の命令鍵ｋ４１を知らない者はこのデータ鍵ｋ４２を共有ライブラリ４から取り出すことはできない。秘匿すべきデータを暗号化するには、このデータ鍵ｋ４２の値と、暗号化するべきメモリ領域のアドレスを、鍵値テーブル１３に格納する。すると指定されたメモリ領域に対して読み書きされるデータは、コード・データ暗号／復号処理部１１により暗号化／復号化される。また、このデータ鍵ｋ４２を使って予め暗号化したデータを共有ライブラリ４の初期化済みデータ４３として配布し、共有ライブラリ４のサブルーチン４２ｃ$_{i(1 \leqq i \leqq n)}$がこれを上記のように復号化して使っても良い。
【００５７】
データを上述のメモリ領域に書き込んでいったん呼出元に復帰した後、再び呼び出されたときにこのデータを読み出す必要がある場合は、同じデータ鍵ｋ４２を鍵値テーブル１３に再度格納すれば良い。
【００５８】
ただし上記の方法では、同じ共有ライブラリ４を使う限り、何回ロードしても毎回同じデータ鍵ｋ４２の値を使うことになる。つまり、同じデータ鍵ｋ４２を使うと、１回のプログラム実行時の暗号化されたデータを保存しておいて、もう１度プログラムを実行した時にこのデータを同じメモリ領域に書き込み、前回実行したときの状態を再現することができる。このようにデータの再利用ができてしまうと、共有ライブラリ４の動作に対する攻撃手段として使われる恐れがあり、不都合な場合がある。
【００５９】
そのような場合は、白川他による特願2000-333635に記載のように、命令鍵とデータ鍵を鍵のペアとみなし、データ鍵ｋ４２としてプロセッサの乱数生成部１５で生成した乱数を使用すればよい。共有ライブラリ４のタスクは、実行時に乱数生成部１５より乱数を取得し、これをデータ鍵ｋ４２として、このデータ鍵ｋ４２を使用して暗号化するべきメモリ領域のアドレスとともに、鍵値テーブル１３に格納する。共有ライブラリ４のタスク毎に乱数生成部１５で生成したデータ鍵ｋ４２を使用すれば、異なるタスクでは異なる値のデータ鍵ｋ４２が使われる。従って、上記のようにして得られたデータ鍵ｋ４２を使って暗号化されたデータは、再利用することができなくなる。
【００６０】
命令鍵ｋ４１とデータ鍵ｋ４２のペアは呼出元に復帰しても内容が失われないため、再び共有ライブラリ４のルーチンが呼び出されても、このデータ鍵ｋ４２で暗号化したデータは読み出すことができる。
【００６１】
（データの受渡し）
共有ライブラリ４とその呼出元（プログラムないしは、その他の共有ライブラリ）は、上記のサブルーチン呼出パラメータブロック８を用いて互いにデータをやりとりすることができる。しかし、このサブルーチン呼出パラメータブロック８の内容は暗号化されていない。そのため、オペレーティングシステムにより内容が見られてしまう恐れがある。

【００６２】
共有ライブラリ４とその呼出元が秘匿データを互いにやりとりするためには、寺本他による特願2000-402672で提案されているように、両者の間でDiffie-Hellman鍵交換シーケンスを用いて鍵の生成を行なえばよい。Diffie-Hellman 鍵交換シーケンスにより生成される鍵は、鍵交換を行なった２者のみが計算できる。すなわち、両者のやりとりを観察することのできるOSであっても、この鍵の値を知ることはできない。以下、この鍵を共通鍵ｃｋと呼ぶことにする。
【００６３】
秘匿データ受渡しのための領域は、OSが提供するメモリ共有手段により、両者の間で共有されるメモリ領域を割り当てる。この領域に対し、共有ライブラリ４と呼出元がそれぞれのタスクの鍵値テーブルエントリ１３０$_{i(0 \leqq i \leqq n)}$に共通鍵ｃｋを登録すれば、相手がこのメモリ領域に暗号化して書き込んだ内容を互いに復号化して読むことができる。このようにして用意したメモリ領域を、以下では共有暗号化データ領域と呼ぶ。
【００６４】
共有ライブラリ４やその呼出元が相互に、相手が意図したプログラム３又は共有ライブラリ４であることを認証する必要がある場合は、上記の鍵交換シーケンスのメッセージに公開鍵方式による署名を与えることにより行う。例えば、プログラム３が共有ライブラリ４を認証する必要がある場合は、共有ライブラリ４に公開鍵と秘密鍵の組を予め与えておき、プログラム３の作成者に共有ライブラリ４の公開鍵を配布しておく。認証は、鍵交換のために共有ライブラリ４がプログラム３に送るメッセージに、共有ライブラリ４の秘密鍵による署名を付加し、プログラム３が共有ライブラリ４の公開鍵を用いてこの署名を検証する。反対に共有ライブラリ４がプログラム３を認証する必要がある場合は、プログラム３に公開鍵と秘密鍵のペアを与えておき、共有ライブラリ４の作成者にプログラム３の公開鍵を配布しておく。認証は、鍵交換のためにプログラムが共有ライブラリ４に送るメッセージに、プログラム３の秘密鍵による署名を付加し、共有ライブラリ４がプログラム３の公開鍵を用いてこの署名を検証する。なお、これらの方式では認証を鍵交換と同時に行うが、もしこれを別々に行なってしまうと、認証をしている相手と鍵交換をしている相手が同一であることを確かめる方法がないため、悪意のあるプログラムによる相手のなりすましが可能になってしまい、認証が正しく行なわれない。
【００６５】
（共有ライブラリの動作）
以上で説明した各部位の基本動作を用いて、本発明の一実施形態における実際の共有ライブラリ４の動作を説明する。ここで説明するプログラム３が、共有ライブラリ４を使用する手順における、共有ライブラリ４側の動作を図９に示す。またその時の、プログラム３側の動作を図１０に示す。なお、図１１は、そのときのプログラム３と共有ライブラリ４それぞれのメモリアドレス空間の配置である。
【００６６】
ここで説明する共有ライブラリ４は、プログラム３からデータを受け取ってある処理を行ないその結果を返す、という動作のルーチンを提供する。その他に、Diffie-Hellman鍵交換シーケンスを行うための補助的なルーチンもプログラムから利用可能である。また、プログラム３の側から共有ライブラリ４を認証することができるように、共有ライブラリ４には前述の認証用の公開鍵と秘密鍵が与えられているものとする。これらのうち、公開鍵の方は共有ライブラリ４と共に配布され、この共有ライブラリ４を使いたいプログラマは、この公開鍵を自分が作るプログラムに組み込むことができる。
【００６７】
共有ライブラリ４を使用するプログラム３が実行されると、先に図６を用いて説明したように、共有ライブラリ４のロードが行なわれる（ステップ９１）。ロードの際に共有ライブラリ４のブートストラップルーチン４２ａ（図６参照）が実行される。この際マイクロプロセッサ１の乱数生成部１５より乱数を取得してこれよりデータ鍵ｋ４２を生成し、これを鍵値テーブル１３に共有ライブラリ４のデータ鍵ｋ４２として格納する。ロードが完

了すると共有ライブラリ４は待機状態になり、プログラム３の実行が開始される。プログラム３も共有ライブラリ４と同様、自分が使うメモリ領域を暗号化するために乱数生成部１５より乱数を取得し、データ鍵ｋ３２を生成して鍵値テーブル１３にデータ鍵ｋ３２として格納する（ステップ１０１）。

【００６８】

プログラム３は共有ライブラリ４の機能を使用する前に、データの受渡しのための共有暗号化データ領域１１５を準備する（ステップ１０２）。次に共有ライブラリ４のDiffie-Hellman鍵交換シーケンスを行うためのルーチンを呼び出す（ステップ１０３）。またこれに伴い、共有ライブラリ４側もDiffie-Hellman鍵交換シーケンスを実行する（ステップ９２）。鍵交換の際、共有ライブラリ４からプログラム３に送るメッセージには、共有ライブラリ４の認証用の秘密鍵を用いて署名を付加し、この認証情報をプログラムへ送付する（ステップ９３）。プログラム３は共有ライブラリ４からこの認証情報を受け取る（ステップ１０４）。ここで、共有ライブラリ４の認証用の公開鍵を用いてこの署名の検証を行う（ステップ１０５）。この検証が成功した場合にのみ鍵交換シーケンスを終了させる（ステップ１０６）。この署名の検証により、鍵交換を行なっている相手の共有ライブラリ４が、意図した正しい共有ライブラリ４であることを確かめることができる。

【００６９】

鍵交換の結果生成される共通鍵ｃｋの値は、プログラム３の側ではプログラム３のデータ鍵ｋ３２により暗号化されてプログラム３以外から読み出せない秘匿データ領域１１２に書き込まれる。また同様に共有ライブラリ４の側でも上記共通鍵ｃｋは、共有ライブラリ４のデータ鍵ｋ４２により暗号化され、共有ライブラリ４以外から読み出せない秘匿データ領域１１４に書き込まれる（ステップ９４）。次にプログラム３は、データ受渡しのための共有メモリ領域をOSに要求して確保し、共通鍵ｃｋをこの共有メモリ領域に対し適用されるよう鍵値テーブル１３に設定する（ステップ１０７）。また、この共有メモリ領域のアドレスを共有ライブラリ４にも通知し、共有ライブラリ４の側でも共通鍵ｃｋを同じメモリ領域に対して適用されるよう鍵値テーブル１３に設定する。一方、共有ライブラリ４は、この共有メモリ領域のアドレスを受け取り、鍵の設定を行う（ステップ９５）。これによりこのメモリ領域は、共有暗号化データ領域１１５として使用できるようになる。

【００７０】

プログラム３が実際に目的の処理を行う共有ライブラリ４のサブルーチン$42c_{i(1\leqq i\leqq n)}$を呼び出すときは、共有暗号化データ領域１１５に渡すべきデータを入れ、またこのデータのチェックサム１１１を計算してデータに付加する。そして共有ライブラリ４のサブルーチン$42c_{i(1\leqq i\leqq n)}$に呼び出す（ステップ１０９）。この呼出信号を受け取ると（ステップ９６）、共有ライブラリ４のサブルーチン$42c_{i(1\leqq i\leqq n)}$はまずチェックサム１１１を検査する（ステップ９７）。チェックサム１１１がデータ内容と一致しない場合はエラーとして終了する（ステップ９７１）。チェックサム１１１が一致した場合、次に共有暗号化データ領域１１５からデータ及び呼出側からの要求を読みだす（ステップ９８）。この処理の過程で作業用のメモリ領域が必要な場合は、共有暗号化データ領域１１５ではなく、共有ライブラリ４独自のデータ鍵ｋ４２により暗号化されたメモリ領域（作業領域）１１６を作成して使用してもよい（ステップ９９）。次に、このデータに対し要求された処理を行う（ステップ９９０）。要求された処理が完了したら、処理結果を共有暗号化データ領域１１５に格納し、チェックサム１１１を付加した後、プログラム３に復帰する（ステップ９９１）。復帰するとプログラム３はチェックサム１１１を検査する（ステップ１１０、ステップ１１１）。これがデータ内容と一致する場合、返されたデータを使って処理を続行するＳ１１２。一致しない場合は、エラーを返し終了する（ステップ１１３）。

【００７１】

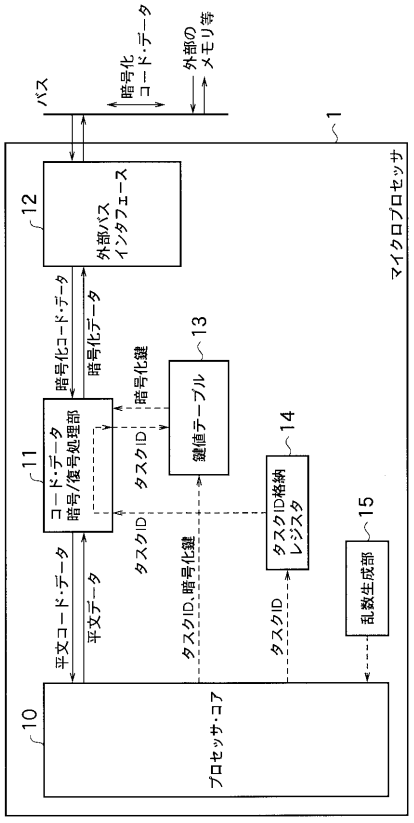
以上で説明した共有ライブラリ４は、鍵交換のための手続きやデータ受渡しの形式が公開されていれば誰でもこれを使うプログラム３を作ることができる。この共有ライブラリ４が安全性に関する要求を満たすことは次のように確認できる。

【００７２】
プログラム３から渡されたデータの秘匿性は、プログラム３と共有ライブラリ４の間で共有される共有暗号化データ領域１１５が、共通鍵ｃｋによって暗号化されていることにより保証される。
【００７３】
ここで暗号化に用いている共通鍵ｃｋはDiffie-Hellman鍵交換により生成している。従って、どちらかが意図的に、あるいは誤ってこの共通鍵ｃｋを開示してしまわない限り、この二者以外に共通鍵ｃｋの値が知られることはない。また、処理結果を返す時も、同じ共有暗号化データ領域１１５を使うため、この内容は、プログラム３および共有ライブラリのみが閲覧可能である。もし処理結果を他のプログラムが読もうとしても、元のDiffie-Hellman 鍵交換による共通鍵ｃｋの値を知らないので、復号化することはできない。
【００７４】
共有ライブラリ４が処理を行なっている途中で、OSが呼出元のプログラム３を別のプログラム３にすり替えて、置き換えた後のプログラム３が共有ライブラリ４から返された処理結果を盗み取るという攻撃が考えられる。しかし、呼出元のプログラムは、すり替えの前後で違う命令鍵ｋ１（ｋ３１）のプログラム３になるため、鍵値テーブル１３でペアとして管理されている命令鍵ｋ１とデータ鍵ｋ２（ｋ３２）の内容は共に変化する。共通鍵ｃｋはすり替え前のデータ鍵ｋ２（ｋ３２）で暗号化された秘匿データ領域１１５に格納されているが、すり替え後は鍵値テーブル１３にあるデータ鍵ｋ２（ｋ３２）の値が変わってしまうため、すり替え後のプログラム３は共通鍵ｃｋを読むことができない。このため共有ライブラリ４より返されたデータをすり替え後のプログラム３が復号化することは原理的に不可能である。
【００７５】
別の攻撃手段として、プログラム３と共有ライブラリ４との間のデータ受渡しの途中過程で、他のプログラム３がこの内容を改変することにより誤動作を誘導することが考えられる。しかし、改変をする側のプログラム３はこのデータの暗号化に使われている共通鍵ｃｋの値を知らないため、改変結果を復号化した結果を予測することはできない。改変を行うとデータを復号化した結果はでたらめな内容になり、チェックサム１１１による整合性の検査で改変が検出される。
【００７６】
複数のタスクが同時に同じ共有ライブラリ４を使用している場合でも、各呼出元のタスク毎に別のタスクを共有ライブラリ４のために割り当てるので、各共有ライブラリ４はそれぞれ別のデータ鍵ｋ４２を使うことになる。このため、共有ライブラリ４がある呼出元から受け取ったデータを、誤って他の呼出元に渡してしまうようなことは発生しない。
【００７７】
共有ライブラリ４を呼び出すプログラム３自体が、共有ライブラリ４の実行コード４２を見ることは、共有ライブラリ４がそれに固有の命令鍵ｋ４１により暗号化されているため、不可能である。また、共有ライブラリ４が処理の過程で作業用に使うメモリの内容も、共有ライブラリ４にしか知られないデータ鍵ｋ４２により暗号化された秘匿データ領域１１４を使うため、これを覗き見ることはできない。
【００７８】
なお、上記の共有ライブラリ４は予め認証用の秘密鍵を持っている。このため、この秘密鍵を知らない第三者は、この共有ライブラリ４の代替となる共有ライブラリ４を勝手に作ることはできない。これは動的リンク共有ライブラリの利点である、既存の共有ライブラリと互換性のある共有ライブラリを自由に作ることができるという特徴に反する。しかし、これは共有ライブラリ４の動作の安全性をプログラム３に対して保証する上で必要不可欠な制約事項である。もし共有ライブラリ４の、元々の提供者以外の開発者が、機能追加等のためにこの共有ライブラリ４と互換性のある新しい共有ライブラリ４を作成する必要がある場合は、元の共有ライブラリ４の提供者に新しい共有ライブラリ４の安全性を確認してもらい、秘密鍵を埋め込んでもらうよう依頼すれば良い。

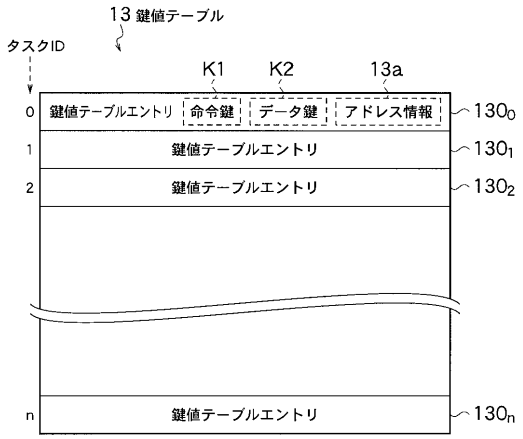
【００７９】
【発明の効果】
上記説明したように、本発明によれば耐タンパマイクロプロセッサ上で動作する、保護されたアプリケーションプログラムから、保護された共有ライブラリを使用することが可能となる。共有ライブラリを使うことにより、プログラムの開発効率や機能拡張性の向上を得ることができる。また、秘匿されたデータのやりとりや、相互認証を可能とすることにより、暗号化されたプログラム
や共有ライブラリ内の処理方法や、処理結果などのデータの秘密を守ることが可能である。
【図面の簡単な説明】
【図１】本発明による共有ライブラリ使用の前提となるマイクロプロセッサの構成図。
【図２】図１のマイクロプロセッサ内部における、鍵値テーブルの構成図。
【図３】アプリケーションプログラムの構成図。
【図４】共有ライブラリの構成図。
【図５】 本発明によるタスク実行開始の手順を示すフローチャート。
【図６】本発明による共有ライブラリをロードする手順を示すフローチャート。
【図７】本発明による共有ライブラリのルーチン呼出手続きを示すシーケンス図。
【図８】本発明によるサブルーチン呼出パラメータブロックの構成図。
【図９】プログラムから共有ライブラリを使用する際の、共有ライブラリ側のフローチャート。
【図１０】プログラムから共有ライブラリを使用する際の、プログラム側のフローチャート。
【図１１】プログラムと共有ライブラリのメモリアドレス空間の配置図。
【符号の説明】
１　耐タンパマイクロプロセッサ
１０　プロセッサ・コア
１１　コード・データ暗号/復号処理部
１２　外部バスインタフェース
１３　鍵値テーブル
１４　タスクＩＤ格納レジスタ
１５　乱数生成部
ｋ１　命令鍵
ｋ２　データ鍵
１３ａ　アドレス情報
１３０_{0}　鍵値テーブルエントリ
１３０_{1}　鍵値テーブルエントリ
１３０_{2}　鍵値テーブルエントリ
１３０_{n}　鍵値テーブルエントリ
３　アプリケーションプログラム
３１　ヘッダ
ｋ３１　命令鍵
３２　実行コード
３３　初期化済みデータ
３４　インポートテーブル
４　　共有ライブラリ
４１　ヘッダ
ｋ４１　命令鍵
４２　実行データ
４２ａ　ブートストラップルーチン
４２ｂ　エントリポイントコード

４２ｃ$_1$　サブルーチン
４２ｃ$_2$　サブルーチン
４２ｃ$_n$　サブルーチン
ｋ４２　データ鍵
４３　初期化済みデータ
４４　インポートテーブル
８　サブルーチン呼出パラメータブロック
８１　共有ライブラリ識別子
８２　サブルーチン識別子
８３　パラメータ
１１２　秘匿データ領域
ｃｋ　共通鍵
１１５　共有暗号化データ領域
１１０　受け渡しデータ
１１１　チェックサム
１１４　秘匿データ領域
１１６　作業領域

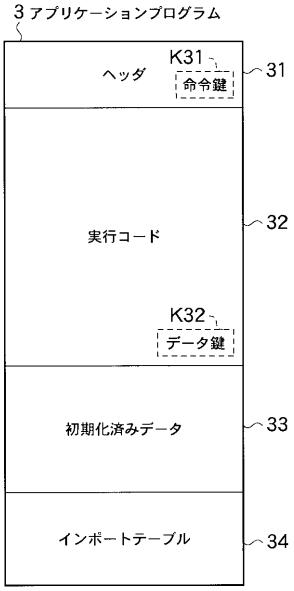
【図１】



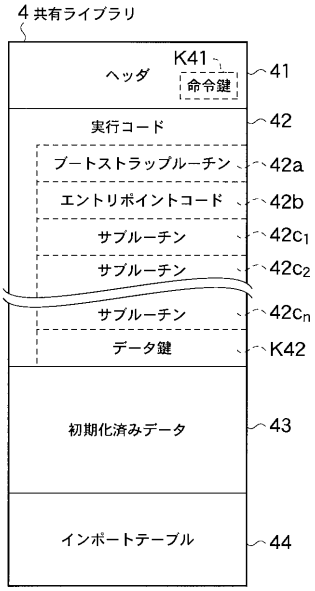
【図２】



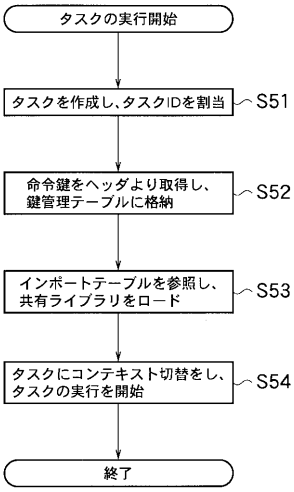
【図３】


3 アプリケーションプログラム
ヘッダ
K31
命令鍵
31
実行コード
32
K32
データ鍵
初期化済みデータ
33
インポートテーブル
34

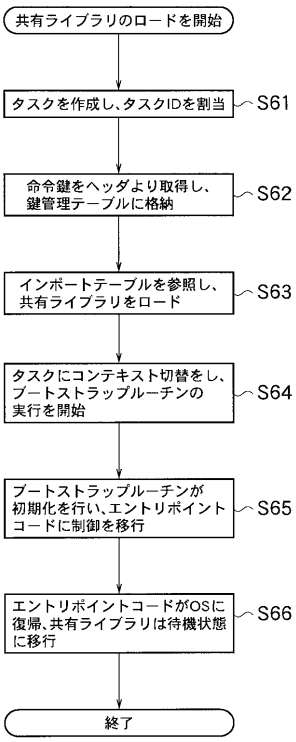
【図４】


4 共有ライブラリ
ヘッダ
K41
命令鍵
41
実行コード
42
ブートストラップルーチン
42a
エントリポイントコード
42b
サブルーチン
42c1
サブルーチン
42c2
サブルーチン
42cn
データ鍵
K42
初期化済みデータ
43
インポートテーブル
44

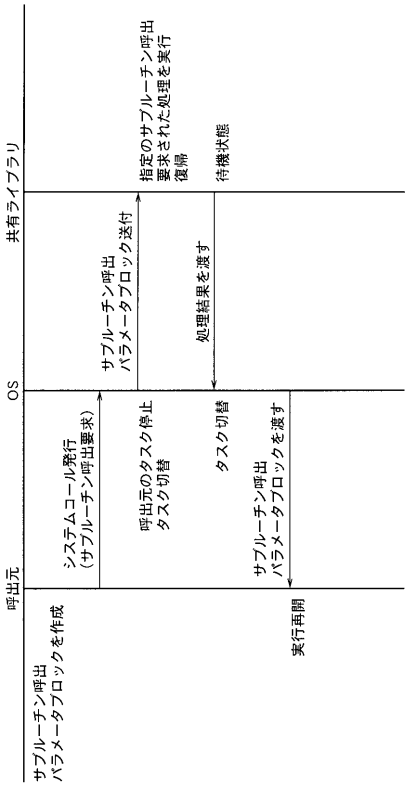
【図５】


タスクの実行開始
タスクを作成し、タスクIDを割当
S51
命令鍵をヘッダより取得し、鍵管理テーブルに格納
S52
インポートテーブルを参照し、共有ライブラリをロード
S53
タスクにコンテキスト切替をし、タスクの実行を開始
S54
終了

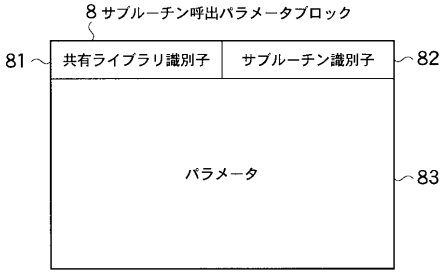
【図６】


共有ライブラリのロードを開始
タスクを作成し、タスクIDを割当
S61
命令鍵をヘッダより取得し、鍵管理テーブルに格納
S62
インポートテーブルを参照し、共有ライブラリをロード
S63
タスクにコンテキスト切替をし、ブートストラップルーチンの実行を開始
S64
ブートストラップルーチンが初期化を行い、エントリポイントコードに制御を移行
S65
エントリポイントコードがOSに復帰、共有ライブラリは待機状態に移行
S66
終了

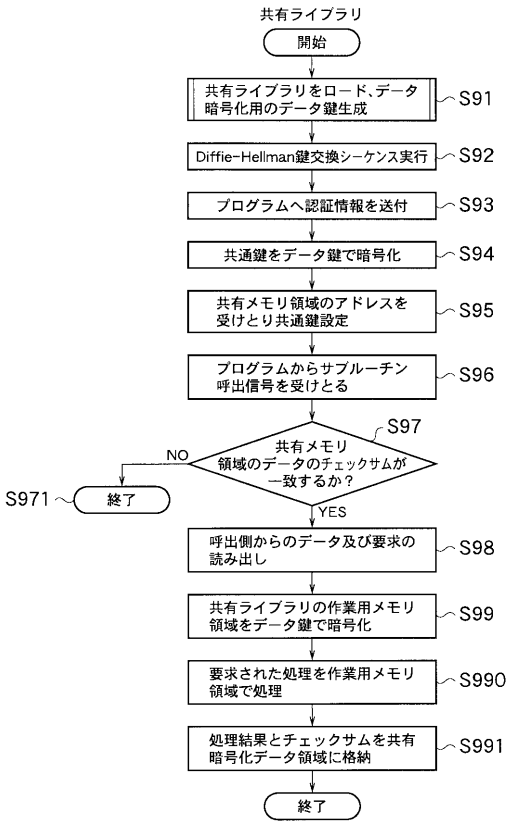
【図７】


共有ライブラリ
OS
呼出元
指定のサブルーチン呼出
要求された処理を実行
復帰
待機状態
サブルーチン呼出
パラメータブロック送付
処理結果を渡す
システムコール発行
(サブルーチン呼出要求)
呼出元のタスク停止
タスク切替
タスク切替
サブルーチン呼出
パラメータブロックを渡す
サブルーチン呼出
パラメータブロックを作成
実行再開

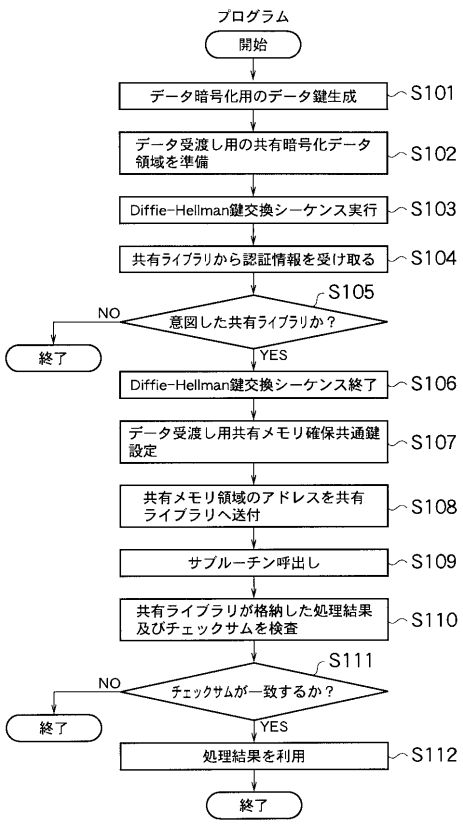
【図８】


8 サブルーチン呼出パラメータブロック
81
共有ライブラリ識別子
サブルーチン識別子
82
パラメータ
83

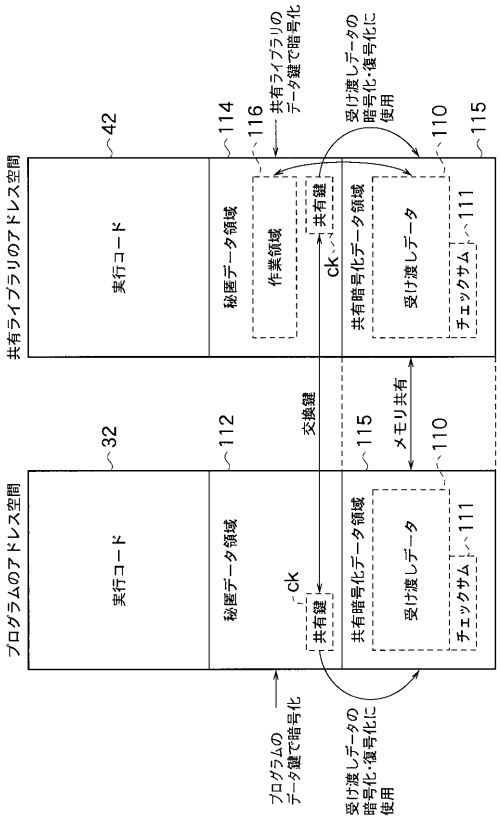
【図９】


共有ライブラリ
開始
共有ライブラリをロード、データ暗号化用のデータ鍵生成
S91
Diffie-Hellman鍵交換シーケンス実行
S92
プログラムへ認証情報を送付
S93
共通鍵をデータ鍵で暗号化
S94
共有メモリ領域のアドレスを受けとり共通鍵設定
S95
プログラムからサブルーチン呼出信号を受けとる
S96
S97
共有メモリ領域のデータのチェックサムが一致するか？
NO
S971
終了
YES
呼出側からのデータ及び要求の読み出し
S98
共有ライブラリの作業用メモリ領域をデータ鍵で暗号化
S99
要求された処理を作業用メモリ領域で処理
S990
処理結果とチェックサムを共有暗号化データ領域に格納
S991
終了

【図１０】


プログラム
開始
データ暗号化用のデータ鍵生成
S101
データ受渡し用の共有暗号化データ領域を準備
S102
Diffie-Hellman鍵交換シーケンス実行
S103
共有ライブラリから認証情報を受け取る
S104
S105
意図した共有ライブラリか？
NO
終了
YES
Diffie-Hellman鍵交換シーケンス終了
S106
データ受渡し用共有メモリ確保共通鍵設定
S107
共有メモリ領域のアドレスを共有ライブラリへ送付
S108
サブルーチン呼出し
S109
共有ライブラリが格納した処理結果及びチェックサムを検査
S110
S111
チェックサムが一致するか？
NO
終了
YES
処理結果を利用
S112
終了

【図１１】


共有ライブラリのアドレス空間
42
実行コード
114
秘匿データ領域
116
作業領域
共有ライブラリのデータ鍵で暗号化
共有鍵
ck
共有鍵
110
115
共有暗号化データ領域
受け渡しデータ
チェックサム
111
受け渡しデータの暗号化・復号化に使用
交換鍵
メモリ共有
プログラムのアドレス空間
32
実行コード
112
秘匿データ領域
ck
共有鍵
115
共有暗号化データ領域
受け渡しデータ
チェックサム
111
110
プログラムのデータ鍵で暗号化
受け渡しデータの暗号化・復号化に使用

フロントページの続き

(51)Int.Cl. F I
H04L 9/00 621A

(74)代理人 100098327
弁理士 高松 俊雄
(72)発明者 山口 健作
神奈川県川崎市幸区小向東芝町１ 株式会社東芝 研究開発センター内
(72)発明者 橋本 幹生
神奈川県川崎市幸区小向東芝町１ 株式会社東芝 研究開発センター内

審査官 石川 正二

(56)参考文献 特開２００２－１４０２３６（ＪＰ，Ａ）
特開２００２－０３２２６８（ＪＰ，Ａ）
特開平１１－１１０１９３（ＪＰ，Ａ）
David Lie, Chandramohan A. Thekkath, Mark Horowitz, "Separating Protection and Resource Management in Operating Systems", [online], 米国, Stanford University, ２００２年５月１７日, [retrieved on 2006.11.13], Retrieved from the Internet, ＵＲＬ, http://mos.stanford.edu/papers/dl_osdi_02.pdf

(58)調査した分野(Int.Cl., ＤＢ名)
G06F 21/22
G06F 11/10
G06F 21/24
H04L 9/10