wolfSSLの ITRON OSサポート

後　援：TOPPERSプロジェクト

2021年3月24日

wolfSSL Japan

Seattle, WA ： Bozeman, MT ：Portland, OR ： Rescue, CA ： Tokyo, JP ： Brisbane, AU

プロフィール

古城 隆 (こじょう たかし)

wolfSSL Japan合同会社、技術統括

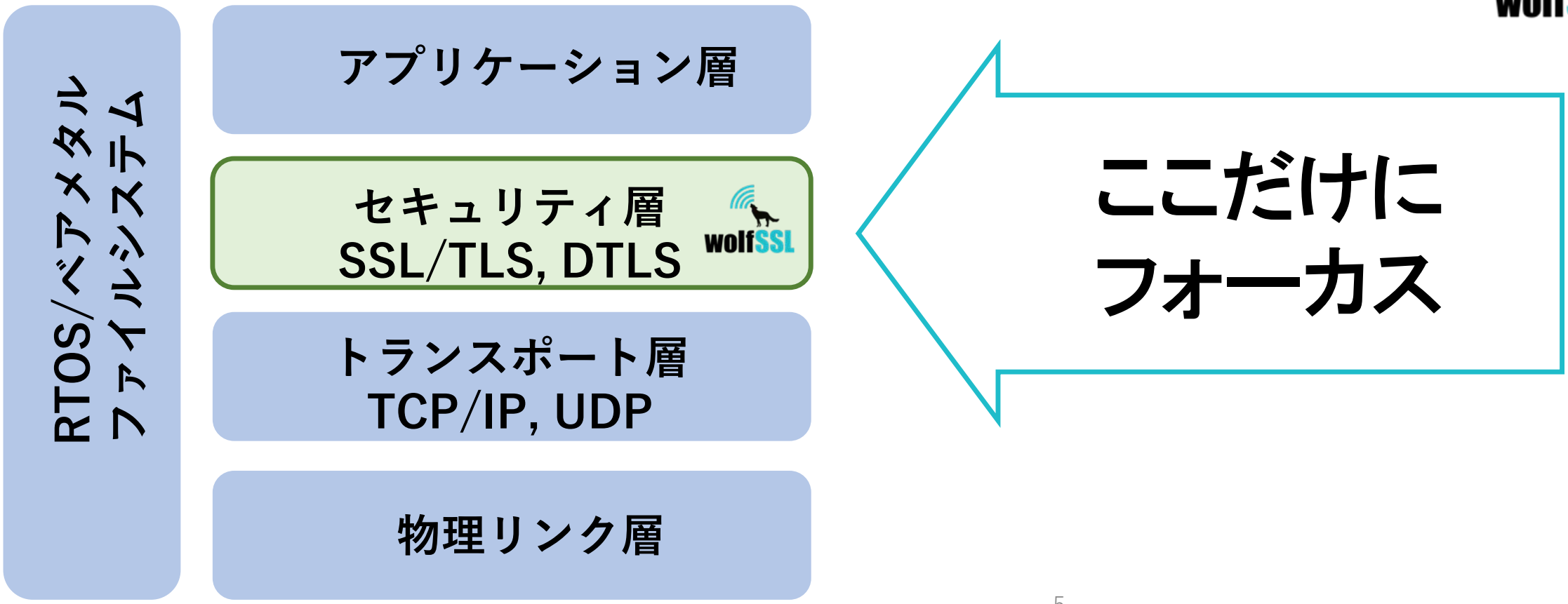
技術責任者業務の傍ら、セキュリティプロトコル、ファームウェア更新をはじめとする最新IoTセキュリティについての執筆活動、各地での講演など。

ソフトウェアエンジニアとして通信系システム、組込み向けRTOS開発、アーキテクチャデザイン、ネットワークベンチャーのCTOなどを経て、2012年より現職。

今日の内容

- wolfSSLの概要
- ITRONとプラットフォーム依存部
  - RTOSカーネル(タスク、排他制御)
  - TCP/IPメッセージング
  - ファイルシステム
  - 実時間時計
  - ヒープ領域管理
  - C言語標準関数
- ビルド環境
  - Make, gcc環境
  - IDE環境
- デバッグサポート

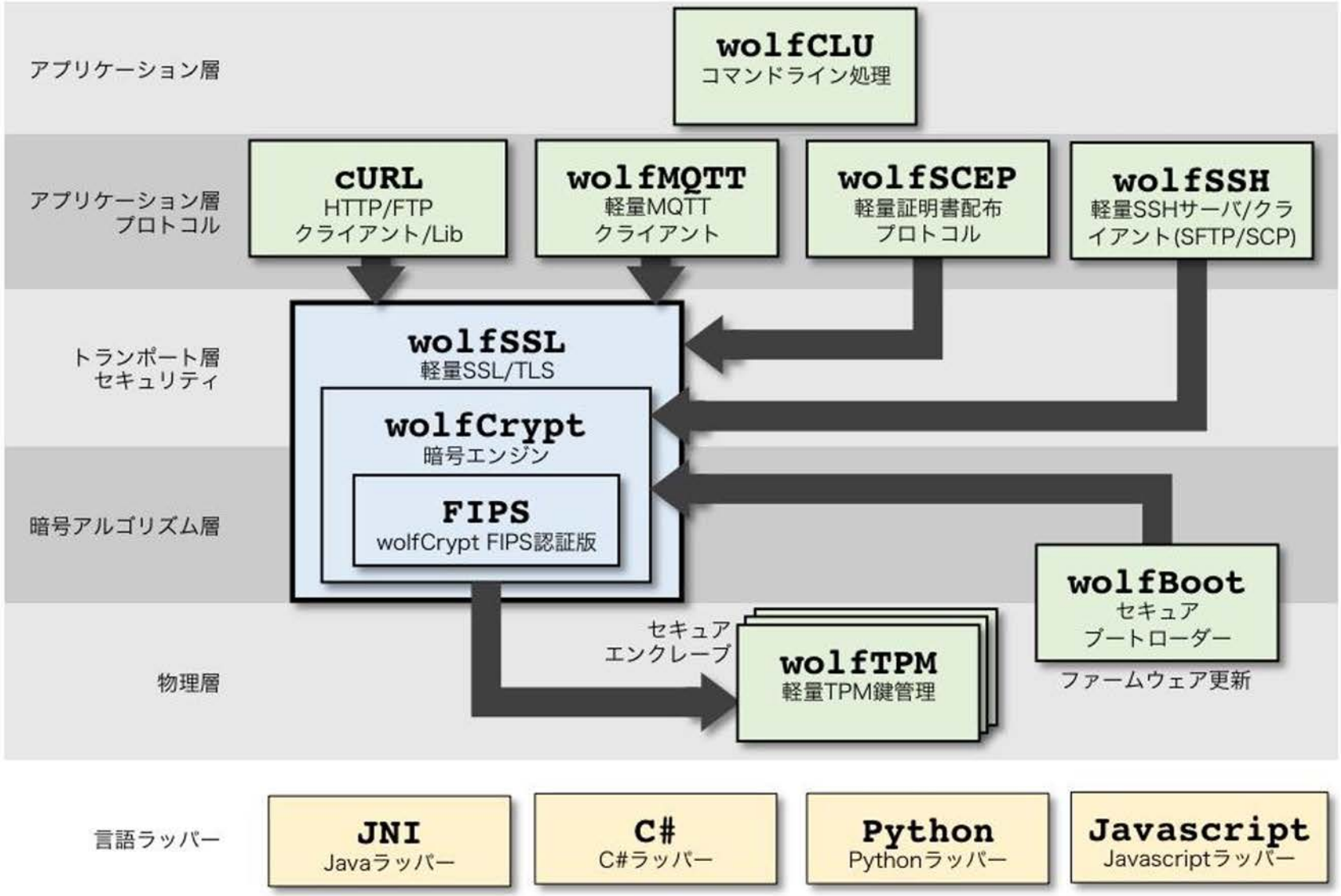
wolfSSLは…



ネットワークセキュリテイ、暗号技術の専門チームです

本社は米国シアトル

wolfSSLの製品


アプリケーション層
wolfCLU
コマンドライン処理
アプリケーション層
プロトコル
cURL
HTTP/FTP
クライアント/Lib
wolfMQTT
軽量MQTT
クライアント
wolfSCEP
軽量証明書配布
プロトコル
wolfSSH
軽量SSHサーバ/クライアント(SFTP/SCP)
トランポート層
セキュリティ
wolfSSL
軽量SSL/TLS
暗号アルゴリズム層
wolfCrypt
暗号エンジン
FIPS
wolfCrypt FIPS認証版
wolfBoot
セキュア
ブートローダー
ファームウェア更新
物理層
セキュア
エンクレーブ
wolfTPM
軽量TPM鍵管理
言語ラッパー
JNI
Javaラッパー
C#
C#ラッパー
Python
Pythonラッパー
Javascript
Javascriptラッパー

世界中のお客さまに
安全な通信をお届けしています

適用事例

スマート工場

病院：検査機器

車載：インフォティメント

電力：スマードグリッド

署名用ペンタブレット

オフィス機器

写真はイメージです

適用事例：ホーム

写真はイメージです

wolfSSLのフォーカス分野

- 高い信頼性
- 長期間安定した運用サポート

産業/ビジネス機器：
- 金融決済端末
- スマートファクトリ
- FA機器、産業用ロボット
- 電力監視
- ビル管理
- 複合機、プリンタ
- アプライアンスボックス

医療機器:
- 内視鏡
- CT検査装置
- 見守りベッド

自動車/鉄道関係:
- 車載AV
- エンジン制御
- 車両テスト/計測機器
- 配車システム
- 信号システム

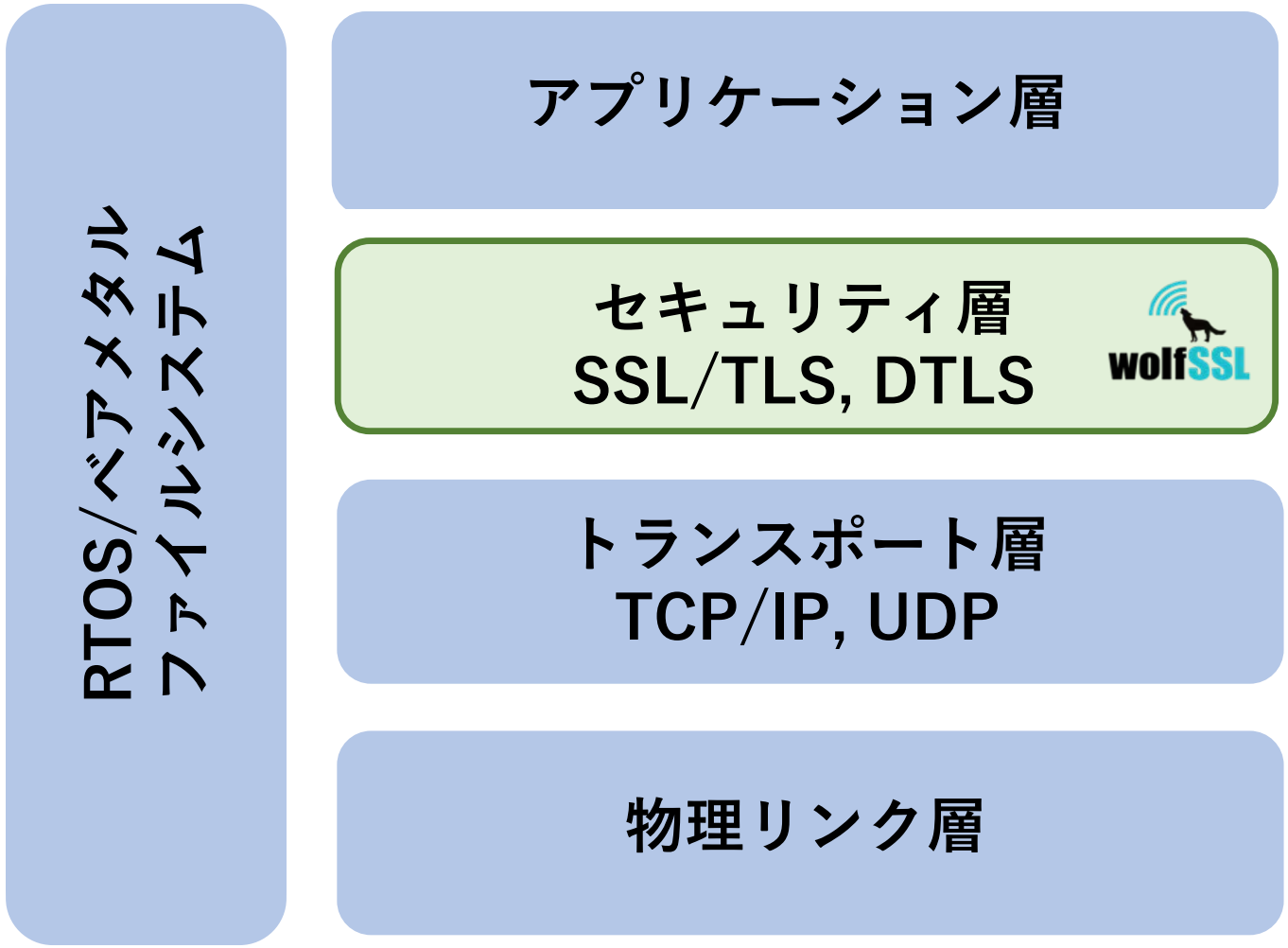
家庭/一般機器:
- デジタルカメラ
- 掃除ロボット
- 健康機器
- スマートライト
- ドアホン/キー
- AV機器
- 太陽光発電
- パソコンソフト
- スマホアプリ/ゲーム

写真はイメージです

今日の内容

- wolfSSLの概要
- ITRONとプラットフォーム依存部
  - RTOSカーネル(タスク、排他制御)
  - TCP/IPメッセージング
  - ファイルシステム
  - 実時間時計
  - ヒープ領域管理
  - C言語標準関数
- ビルド環境
  - Make, gcc環境
  - IDE環境
- デバッグサポート

セキュリティ層にフォーカス



wolfSSLのまとめ

軽量 SSL / TLS ライブラリー

軽量、ポータブル、C言語ベース

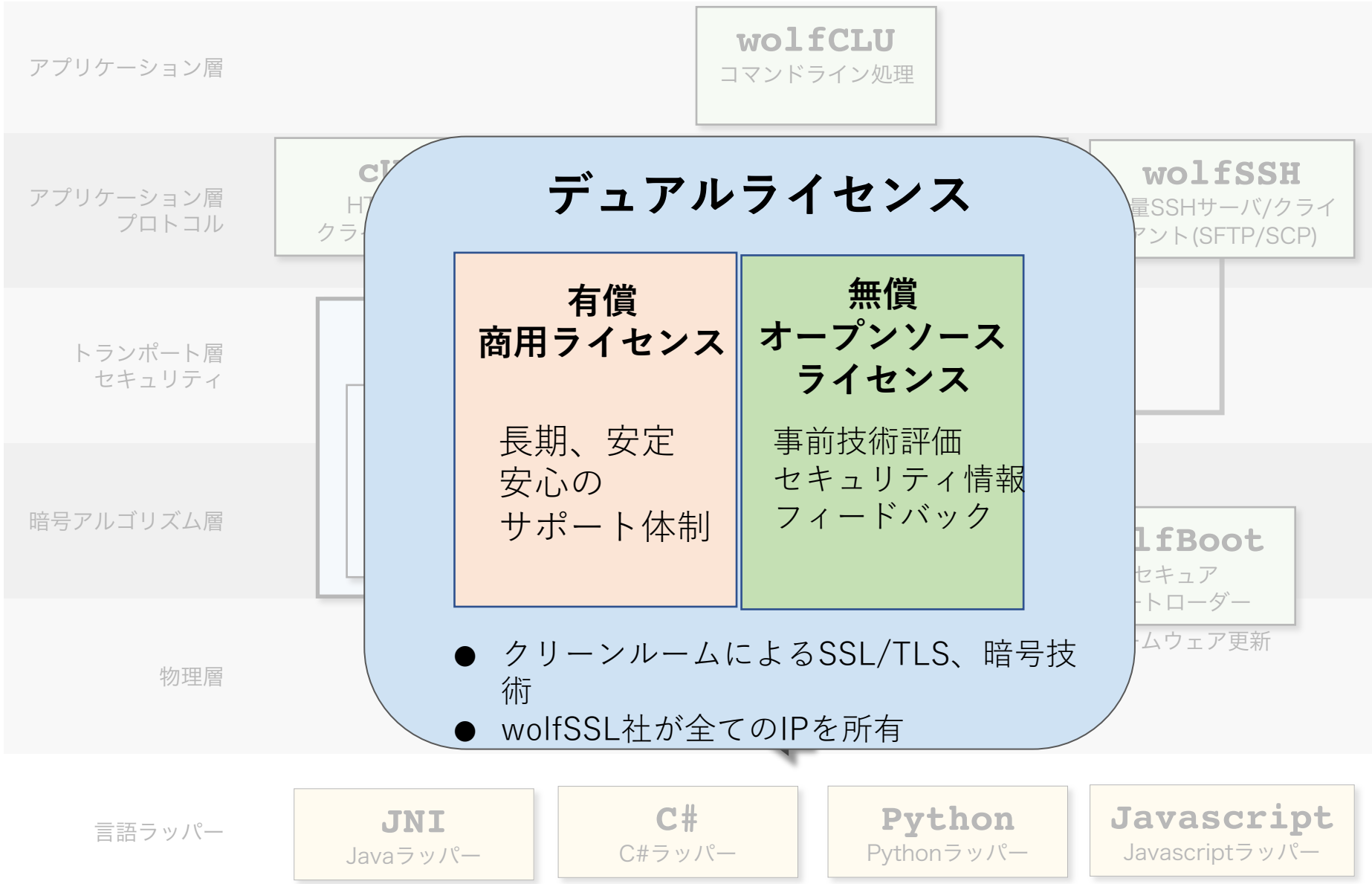
- ✓TLS 1.2、TLS1.3、DTLS 1.2対応
- ✓ROM: 20-100 kB
- ✓RAM: セッションあたり1-36 kB
- ✓OpenSSLに比べて、最小20分の1
- ✓多数のOSサポート

Windows, Linux, Mac OS X, Solaris, ThreadX, VxWorks, FreeBSD, NetBSD, OpenBSD, embedded Linux, WinCE, Haiku, OpenWRT, Nintendo Wii and Gamecube through DevKitPro

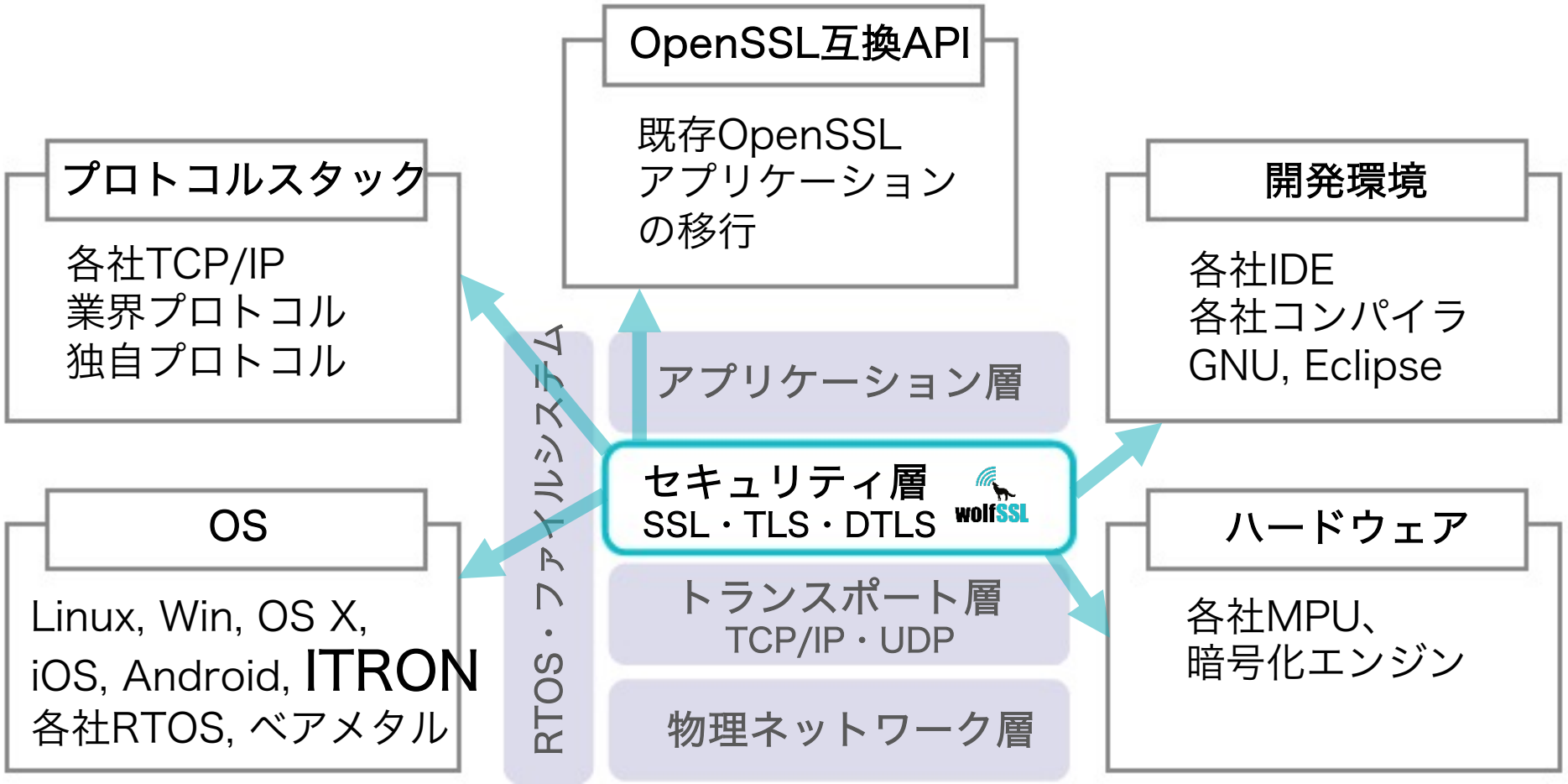
iPhone (iOS), Android, QNX, MontaVista, Nucleus, NonStop, **TOPPERS/ASP uITRON, uT-Kernel, T-Kernel**, Micrium uC/OS-III, FreeRTOS, SafeRTOS, Freescale MQX

TinyOS, HP/UX, ARC MQX, TI-RTOS, uTasker, embOS, INtime, Mbed, RIOT, CMSIS-RTOS, FROSTED, Green Hills INTEGRITY, Keil RTX, TOPPERS, PetaLinux, Apache Mynewt

wolfSSLの製品


アプリケーション層
wolfCLU
コマンドライン処理
アプリケーション層
プロトコル
wolfSSH
量SSHサーバ/クライ
アント(SFTP/SCP)
トランポート層
セキュリティ
暗号アルゴリズム層
lfBoot
セキュア
トローダー
ムウェア更新
物理層
デュアルライセンス
有償
商用ライセンス
長期、安定
安心の
サポート体制
無償
オープンソース
ライセンス
事前技術評価
セキュリティ情報
フィードバック
● クリーンルームによるSSL/TLS、暗号技術
● wolfSSL社が全てのIPを所有
言語ラッパー
JNI
Javaラッパー
C#
C#ラッパー
Python
Pythonラッパー
Javascript
Javascriptラッパー

プラットフォームとの親和性



ITRON系カーネル

- TOPPERS/ASP3
  フルビルド
  Base Platform
  ASP, lwIp, デバイスドライバー
  ←評価ボードでクイックスタート
  オープンソース

- TOPPERSベース
  各社自製RTOS
  独自カスタマイズ

- μITRON4.0
  eForce μC3、ミスポNORTi
- T-Kernel
  eSol eT-Kernel

上位レイヤー、他コンポーネントを含めた手厚いベンダーサポート

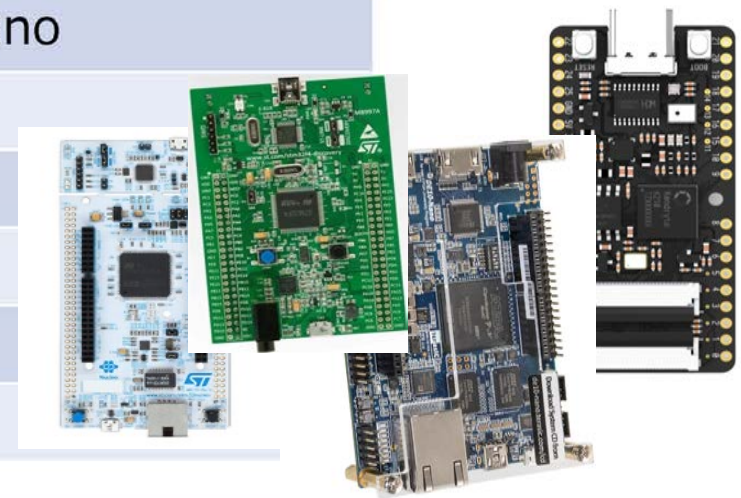
TOPPERS BASE PLATFORMとは

- TOPPERSのワーキンググループの一つ
  教育WGで作成した教育コンテンツで使う
  ソフトウェアプラットフォーム
- 教育コンテンツは組込み技術者養成用
  - 基礎１、２、３のコースがありそれぞれ２日間
    - 基礎1：ハードウェア、開発環境、デバドラの開発実習
    - 基礎2：RTOSの導入、RTOS運用、TOPPERS BASE PLATFORM(ST)環境セットアップ
    - 基礎3：SPI,ADCデバイスドライバ、グラフィックLCDの表示方法、SDカードを使ったFATファイルの理解と学習
  - とっても詳しいコンテンツです！

対応ハードウェア

- コンテンツではSTマイクロの「NUCLEO-F401RE」または「NUCLEO-F446RE」を使用
- その他のSTM32やRISC-V、 Intel Cyclone V SoC向けのソフトウェアが公開されている
- 一部紹介

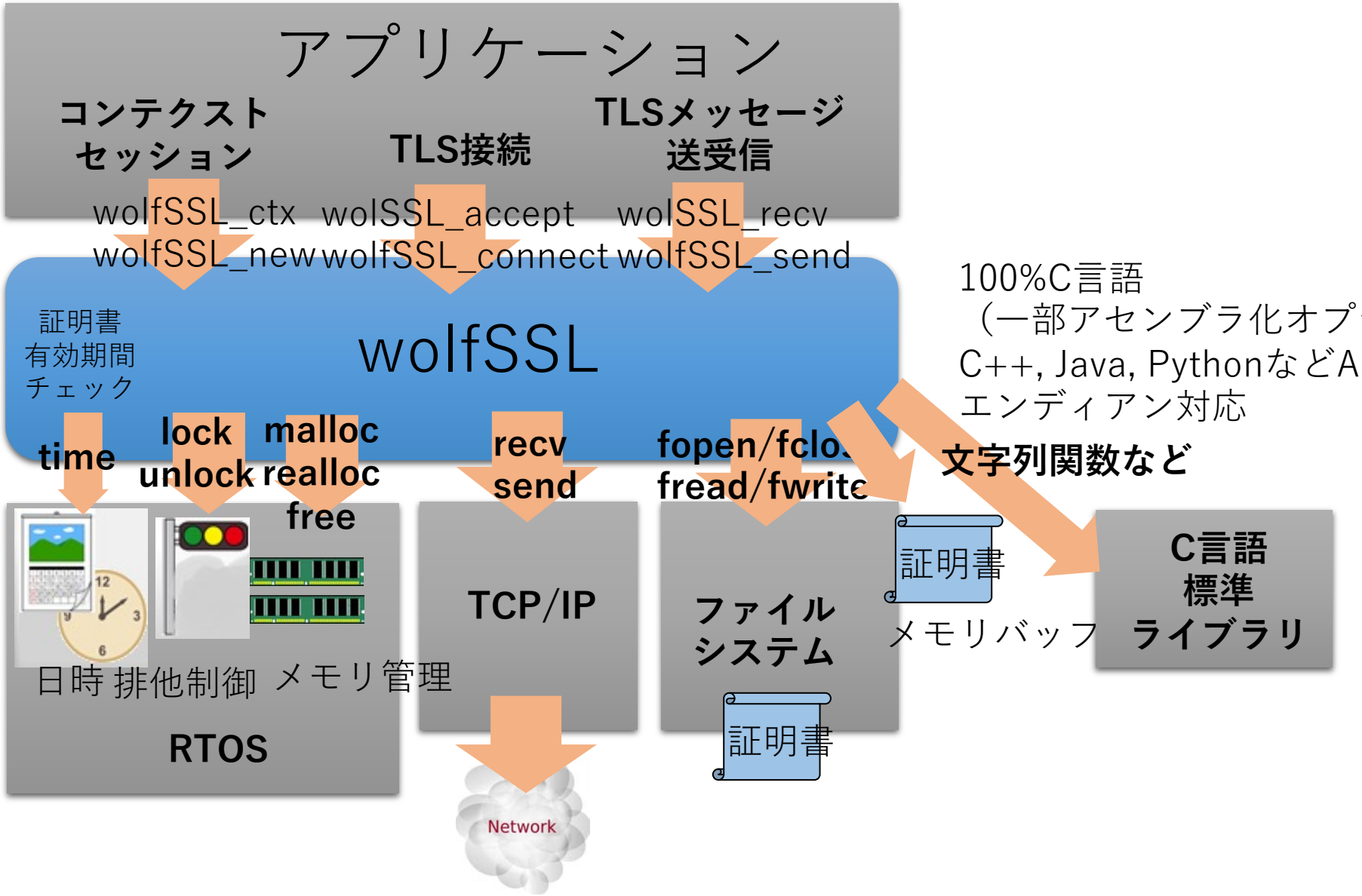
CPU	ボード
ARM Cortex-A9	Cyclone V SoC DE10-Nano
ARM Cortex-M7	STM32F7 Discovery
ARM Cortex-M4	STM32F401 Nucleo-64
ARM Cortex-M4/M0	STM32WB55 Nucleo
ARM Cortex-M0+	STM32G071 Nucleo-64
RISC-V K210	Maixシリーズ



さまざまなITRON環境

- **プロセッサ**
  ARM系, Intel系, RISC-V, Microchip, PowerPC, …
- **カーネル**
  TOPPERS/ASP, TOPPERSベース自製、μITRON互換, T-Kernel/μT-Kernel
- **TCP/IP**
  TINET、BSD Socket、またはそのサブセット, lwIp, …
- **ファイルシステム**
  POSIX, 独自ファイルシステム、ファイルシステム無し、メモリーファイルシステム
- **ビルド環境**
  gcc/make, IDE (Eclipse, IAR, MDK, MCU各社製,…)
- **Cコンパイラ、C標準ライブラリ**
  ANSI-C99, 89, 各社組込み向けCコンパイラ

wolfSSLのプラットフォーム依存



100%C言語
（一部アセンブラ化オプション有）
C++, Java, PythonなどAPIラッパー
エンディアン対応

プロセッサ

- アーキテクチャ、命令セットの違いは**Cコンパイラ**で吸収
- **ワードサイズ**

  必須：int 32ビット以上
- **エンディアン**

  デフォルト：リトルエンディアン

  オプション指定：ビッグエンディアン
- **メモリー量**

  スタックサイズ：10kバイト

  ヒープサイズ：6kバイト（ネットワークI/Oバッファ用）

  ただし、v.4.5.0以前は

  スタックサイズ：24kバイト

  ヒープサイズ：32kバイト

  - スタックサイズ縮小オプション　→　ヒープを使用

RTOSカーネル

- **排他制御**
  プラットフォームとなるRTOSのセマフォなどを利用

- **コンフィグオプション**でサポート
  WOLFSSL_uITRON4
  WOLFSSL_uTKERNEL2

- **非互換、互換性に不安のある場合は**
  独自MUTEXオプション：WOLFSSL_USER_MUTEX
  初期化　　：int wc_InitMutex(wolfSSL_Mutex* m) { ... }
  解放　　：int wc_FreeMutex(wolfSSL_Mutex *m) { ... }
  ロック　　：int wc_LockMutex(wolfSSL_Mutex *m) { ... }
  アンロック：int wc_UnLockMutex(wolfSSL_Mutex *m) { ... }

排他制御コードの例

```
int wc_LockMutex(wolfSSL_Mutex* m)
{
      if(wai_sem(m->id) == E_OK)
            return 0;
       else
             return BAD_MUTEX_E;
 }

 int wc_UnLockMutex(wolfSSL_Mutex* m)
 {
      if(sig_sem(m->id) == E_OK)
            return 0;
       else
            return BAD_MUTEX_E;
 }
```

TCP/IPメッセージ送受信

TPCの接続、切断まではライブラリー外、アプリケーション側で適宜

- デフォルトは**BSD Socket**
- **コンフィグオプション**でサポート
  LwIP、ベンダー製品(MQX, freeRTOS TCP, netX, その他)

- 独自API
  - コンフィグオプション：**WOLFSSL_USER_IO**
  - **ユーザ定義のTCPメッセージ送受信コールバック関数**
    実行時にコールバック登録用APIにより登録
    TCP接続時にTCPソケット(またはディスクリプタなど)を登録
  サンプルプログラム：github.com/wolfssl/wolfssl-examples/tls

ファイルシステム

- 主な用途は証明書、鍵ファイル
- デフォルトはPOSIXファイル
- 独自、メモリーファイルなどの場合、マクロ定義

  XFILE
  XFOPEN, XFCLOSE,
  XFSEEK, XFTELL, XREWIND, XSEEK_END
  XFREAD(BUF, SZ, AMT, FD) , XFWRITE(BUF, SZ, AMT, FD)

- ファイルシステム無し
  - 証明書、鍵などメモリーバッファーに格納

  同一機能のファイル用API、メモリーバッファー用APIがサポートされている
  例:プライベート鍵のロード
  メモリーバッファから：wolfSSL_CTX_use_PrivateKey_buffer
  ファイルから　　：wolfSSL_CTX_use_PrivateKey_file

実時間時計

主な用途は証明書の有効期限チェック
　　→　正確な実時間は必要としない

- デフォルト：UNIX Epoch Time　time()
- 独自API指定：USER_TIME
　　通常ハードウェアRTCから取得

ヒープ領域管理

- デフォルト：malloc/free/realloc

- ユーザ独自：
  コンフィグオプション指定：XMALLOC_USER
  #define XMALLOC(size. heap, type)
  #define XFREE(p, heap, type)
  #define XREALLOC(p, size, heap, type)

- wolfSSLのヒープ管理を使用
  ベアメタル、可変長バッファ管理が無い場合など
  アプリケーションで静的プール領域だけ確保し、wolfSSLで管理する
  コンフィグオプション：WOLFSSL_STATIC_MEMORY
  注意：一部、異なる初期化APIを使用する必要あり
  （詳細はAPIレファレンス参照）

C言語標準関数

- 標準I/O
- ストリング

  標準と異なる場合はマクロ定義

TCPプログラムからTLSプログラムへ

- プログラムの全体構造は変更不要

- TLS処理コンテクストの確保とCA証明書などのロード

  コンテクスト確保：wolfSSL_CTX_new
  CA：wolfSSL_CTX_load_verify_locations
  証明書：wolfSSL_CTX_use_certificate_file
  秘密鍵：wolfSSL_use_PrivateKey_file

- TCP接続(connect, accept)のあとにTLS接続追加

  リソース確保：wolfSSL_new
  ハンドシェーク：wolfSSL_connect, wolfSSL_accept

- TCPメッセージ送受信をTLS送受信に

  send, recv → wolfSSL_write, read

サンプルプログラム集: github.com/wolfssl/wolfssl-examples/tls
TCPクライアント、サーバ：tcp_client.c, tpc_server.c
TLSクライアント、サーバ：tls_client.c, tls_server.c

TCPクライアントプログラムの例

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { … }          /* ソケット確保 */
                …
if ((ret = connect(sockfd, (struct sockaddr*) &servAddr, sizeof(servAddr))) { … }          /* TCP接続 */

if (fgets(buff, sizeof(buff), stdin) == NULL) { … }          /* 送信データの準備　*/

if ((ret = send(ssl, buff, len)) != len) { … }          /* TCPメッセージ送信　*/

if ((ret = recv(ssl, buff, sizeof(buff)-1)) == -1) { … }          /* TCPメッセージ受信 */

printf("Server: %s¥n", buff);                                          /* 受信データの処理 */

socket_cleanup:
   close(sockfd);                                                      /* ソケット解放 */
```

TLSクライアントプログラムの例（TLS通信）

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { … }                    /* ソケット確保 */
                …
if ((ret = connect(sockfd, (struct sockaddr*) &servAddr, sizeof(servAddr))) { … }   /* TCP接続 */

if ((ctx = wolfSSL_CTX_new(wolfTLSv1_3_client_method())) == NULL) { … }        /* TLSコンテクスト確保 */
if ((ret = wolfSSL_CTX_load_verify_locations(ctx, CERT_FILE, NULL)) { .. }     /* CA証明書ロード */

if ((ssl = wolfSSL_new(ctx)) == NULL) { … }                                    /* TLSセッション管理の確保 */
if ((ret = wolfSSL_connect(ssl)) != SSL_SUCCESS) {  … }                        /* TLS接続 */

if (fgets(buff, sizeof(buff), stdin) == NULL) { … }                            /* 送信データの準備　*/

if ((ret =  send(ssl, buff, len)) != len) { … }                                /* TCPメッセージ送信　*/

if ((ret = recv(ssl, buff, sizeof(buff)-1)) == -1) { … }                       /* TCPメッセージ受信 */

printf(“Server: %s¥n”, buff);                                                                  /* 受信データの処理 */

cleanup:                                                                                       /* TLSリソース解放 */
   wolfSSL_free(ssl);
ctx_cleanup:
   wolfSSL_CTX_free(ctx);
   wolfSSL_Cleanup();
socket_cleanup:                                                                                /* ソケット解放 */
   close(sockfd);
```

TLSクライアントプログラムの例（TLS通信）

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { … }                    /* ソケット確保 */
                …
if ((ret = connect(sockfd, (struct sockaddr*) &servAddr, sizeof(servAddr))) { … }   /* TCP接続 */

if ((ctx = wolfSSL_CTX_new(wolfTLSv1_3_client_method())) == NULL) { … }        /* TLSコンテクスト確保 */
if ((ret = wolfSSL_CTX_load_verify_locations(ctx, CERT_FILE, NULL)) { .. }     /* CA証明書ロード */

if ((ssl = wolfSSL_new(ctx)) == NULL) { … }                                    /* TLSセッション管理の確保 */
if ((ret = wolfSSL_connect(ssl)) != SSL_SUCCESS) {  … }                        /* TLS接続 */

if (fgets(buff, sizeof(buff), stdin) == NULL) { … }                            /* 送信データの準備　*/

if ((ret =  send(ssl, buff, len)) != len) { … }                                /* TCPメッセージ送信　*/

if ((ret = recv(ssl, buff, sizeof(buff)-1)) == -1) { … }                       /* TCPメッセージ受信 */

printf(“Server: %s¥n”, buff);                                                                  /* 受信データの処理 */

cleanup:                                                                                       /* TLSリソース解放 */
   wolfSSL_free(ssl);
ctx_cleanup:
   wolfSSL_CTX_free(ctx);
   wolfSSL_Cleanup();
socket_cleanup:                                                                                /* ソケット解放 */
   close(sockfd);
```

TLSクライアントプログラムの例（TLS通信）

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { … }                    /* ソケット確保 */
                …
if ((ret = connect(sockfd, (struct sockaddr*) &servAddr, sizeof(servAddr))) { … }   /* TCP接続 */

if ((ctx = wolfSSL_CTX_new(wolfTLSv1_3_client_method())) == NULL) { … }        /* TLSコンテクスト確保 */
if ((ret = wolfSSL_CTX_load_verify_locations(ctx, CERT_FILE, NULL)) { .. }     /* CA証明書ロード */

if ((ssl = wolfSSL_new(ctx)) == NULL) { … }                                    /* TLSセッション管理の確保 */
if ((ret = wolfSSL_connect(ssl)) != SSL_SUCCESS) {  … }                        /* TLS接続 */

if (fgets(buff, sizeof(buff), stdin) == NULL) { … }                            /* 送信データの準備　*/

if ((ret = wolfSSL_write(ssl, buff, len)) != len) { … }                        /* TLSメッセージ送信　*/

if ((ret = wolfSSL_read(ssl, buff, sizeof(buff)-1)) == -1) { … }               /* TLSメッセージ受信 */

printf("Server: %s¥n", buff);                                                          /* 受信データの処理 */

cleanup:                                                                               /* TLSリソース解放 */
    wolfSSL_free(ssl);
ctx_cleanup:
    wolfSSL_CTX_free(ctx);
wolfSSL_Cleanup();
socket_cleanup:                                                                        /* ソケット解放 */
    close(sockfd);
```

TLSクライアントプログラムの例(準備)

```
#include <wolfssl/options.h>    または #include <wolfssl/wolfcrypt/settings.h>
                                                    /* コンフィグオプション・ヘッダー */
#include <wolfssl/ssl.h>                            /* wolfSSLライブラリーヘッダー */

#define CERT_FILE "../certs/ca-cert.pem"

int main(int argc, char** argv) {

 /* declare wolfSSL objects */
    WOLFSSL_CTX* ctx;               /* TLS処理コンテクスト全体の管理ブロック */
    WOLFSSL*     ssl;               /* TLSセッションごとの管理ブロック */

 if ((ret = wolfSSL_Init()) != WOLFSSL_SUCCESS) { … } /* ライブラリ初期化 */
```

ビルド環境：IDE

- ソースファイル登録
  Cソースファイル一式を登録
  TLSプロトコル：<wolfSSLルート>/src/*.c
  暗号アルゴリズム：<wolfSSLルート>/ wolfcrypt/src/*c
  （ただし、src/bio.c, wolfcrypt/src/{misc.c, evp.c}を除く

- インクルードパス登録
  <wolfSSLルート>

- ユーザ・コンフィグレーション
  user_settings.h ヘッダーファイルを**<wolfSSLルート>直下**に
  Pre-definedマクロ名として**WOLFSSL_USER_SETTINGS**を定義

ビルド環境：コマンド

configureまたはcmakeコマンドでMakefileを生成　→ make

$ ./configure オプション CFLAGS=“-Dxxxx … “
$ make　　←ライブラリ作成、各種テスト、サンプルプログラム一式
　　ライブラリ：<wolfSSLルート>/src/.libs/libwolfssl.so
$ sudo make install
　　インストールディレクトリ(usr/local/wolfsslなど)にコピー

configureオプション例：
--enable-static：静的ライブラリー（libwolfssl.a）
--enable-debug: デバッグログ
--help: ヘルプメッセージ

ファイル構成

項　目		ディレクトリ名
プログラム	**SSL/TLS層**	**src/**
	暗号エンジン層	**wolfcrypt/src**
ヘッダーファイル	SSL/TLS層	wolfssl/
	暗号エンジン層	wolfssl/wolfcrypt
テスト		tests/
wolfCryptテストプログラム		wolfcrypt/test
ベンチマークプログラム		wolfcrypt/benchmark
サンプルプログラム		examples/client examples/server その他
IDEプロジェクト、設定ファイル（IAR, MDK ARM, XCODE…）		IDE/
テスト用証明書, 鍵		certs/*.{pem, der} wolfssl/certs_test.h
ユーザ定義オプション		user_settings.h

ビルドオプションの例

configure オプション	ヘッダー ファイル指定	説明
--disable-filesystem	NO_FILESYSTEM	ファイルシステム無し
--enable-dtls	WOLFSSL_DTLS	DTLS
--enable-opensslextra	OPENSSL_EXTRA	OpenSSL互換API
--enable-debug	DEBUG_WOLFSSL	デバッグ機能
--disable-pwdbased	NO_PWDBASED	パスワードベース認証無効化
--enable-aesgcm	HAVE_AESGCM	暗号アルゴリズムAES-GCM
--enable-ecc	HAVE_ECC	楕円曲線暗号
--enable-curve25519	HAVE_CURVE25519	Curve25519楕円曲線暗号
--disable-sha256	NO_SHA256	SHA256ハッシュ関数無効化

製品に含まれるサンプルプログラム

- **テストプログラム**
  <wolfSSLルート>/wolfcrypt/test/test.c
    暗号アルゴリズムのテストプログラム。ターゲット環境での最初のテスト用
- **ベンチマークプログラム**
  暗号アルゴリズム：<wolfSSLルート>/wolfcrypt/test/benchmark.c
  TLS通信：<wolfSSLルート>/examples/benchmark/tsl_bench.c
- **テスト用TLSクライアント、サーバ**
  <wolfSSLルート>/examples/echoclient.c, echoserver.c
- **対向テスト用TLSクライアント、サーバ**
  <wolfSSLルート>/examples/client.c, server.c
  - Linux, PCなどの環境で動作
  - ターゲット側のTLSの対向
  - コマンドオプションにより各種テスト条件

デバッグサポート

早めにヘルプデスク(support@wolfssl.com)にどうぞ！

- **ビルド時のトラブル**
  ビルドエラーメッセージ一式をヘルプデスクに

- **APIのエラーコード**
  戻り値：OK, Fail
  詳細コード：必ずwolfSSL_get_error()で取得
  エラーメッセージ：wolfSSL_ERR_error_string() で取得（メモリーに余裕があれば）
  戻り値、**詳細コード**をヘルプデスクに

- **実行時デバッグ情報**
  ライブラリ内実行状況のログ
  コンフィグオプション：DEBUG_WOLFSSL
  実行時：wolfSSL_Debugging_ON()、wolfSSL_Debugging_OFF()
  デバッグログ、Wireshakによるパケットキャプチャをヘルプデスクに

- **リソース管理**
  - 解放漏れチェック
  コンフィグオプション：WOLFSSL_TRACK_MEMORY

Q&A

ご静聴
ありがとうございました

最新情報はTwitterでもお知らせします。

問合せ: info@wolfssl.jp