Część 2

Ogólne informacje o programowych bibliotekach kryptograficznych

Biblioteki kryptograficzne

Development language

Czym jest biblioteka kryptograficzna?

Jest to API umożliwiający wywoływanie funkcji charakterystycznych dla różnego typu usług odwołujących się do algorytmów i mechanizmów kryptograficznych.

Java od sierpnia 2019

Takich bibliotek mniej lub bardziej uniwersalnych jest wiele.

Także ich dostępność dla potrzeb komercyjnych bywa różna.

D.J.Bernstein, T.Lange, P.Schwabe

Np. obok przedstawiono zestaw bibliotek „open source”; których zakres funkcjonalności porównano na stronie:

https://en.wikipedia.org/wiki/Comparison_of_cryptography_libraries

Implementacja	Język
Botan	C++
Bouncy Castle	C#
cryptlib	C
Crypto++	C++
GnuTLS	C
Libgcrypt	C
libsodium	C
NaCl	C
Nettle	C
Network Security Services (NSS)	C
OpenSSL	C
wolfCrypt	C

RSA BSAFE – kod źródłowy dostępny po zakupie biblioteki od RSA Security

„Przenośność” bibliotek kryptograficznych - przykłady

Implementacja	Wspierane systemy operacyjne
Botan	Linux, Windows, macOS, Android, iOS, FreeBSD, NetBSD, OpenBSD, DragonflyBSD, AIX, QNX, Haiku, IncludeOS
Bouncy Castle	General Java API: J2ME, Java Runtime Environment 1.1+, Android. Java FIPS API: Java Runtime 1.5+, Android. C# API (General & FIPS): CLR 4.
cryptlib	AMX, ARINC 653, BeOS, ChorusOS, CMSIS-RTOS/mbed-rtos, DOS, DOS32, eCOS, embOS, FreeRTOS/OpenRTOS, ultron, MQX, MVS, Nucleus, OS/2, Palm OS, QNX Neutrino, RTEMS, SMX, Tandem NonStop, Telit, ThreadX, uC/OS II, Unix (AIX, FreeBSD, HP-UX, Linux, macOS, Solaris, etc.), VDK, VM/CMS, VxWorks, Win16, Win32, Win64, WinCE/PocketPC/etc, XMK
Crypto++	Unix (AIX, OpenBSD, Linux, MacOS, Solaris, etc.), Win32, Win64, Android, iOS, ARM
Libgcrypt	All 32 and 64 bit Unix Systems (GNU/Linux, FreeBSD, NetBSD, macOS etc.), Win32, Win64, WinCE and more
libsodium	macOS, Linux, OpenBSD, NetBSD, FreeBSD, DragonflyBSD, Android, iOS, 32 and 64-bit Windows (Visual Studio, MinGW, C++ Builder), NativeClient, QNX, JavaScript, AIX, MINIX, Solaris
OpenSSL	Solaris, IRIX, HP-UX, MPE/iX, Tru64, Linux, Android, BSD (OpenBSD, NetBSD, FreeBSD, DragonflyBSD), NextSTEP, QNX, UnixWare, SCO, AIX, 32 and 64-bit Windows (Visual Studio, MinGW, UWIN, CygWin), UEFI, macOS (Darwin), iOS, HURD, VxWorks, uClinux, VMS, DJGPP (DOS), Haiku
wolfCrypt	Win32/64, Linux, macOS, Solaris, ThreadX, VxWorks, FreeBSD, NetBSD, OpenBSD, embedded Linux, WinCE, Haiku, OpenWRT, iPhone (iOS), Android, Nintendo Wii and Gamecube through DevKitPro, QNX, MontaVista, NonStop, TRON/ITRON/µITRON, Micrium's µC/OS, FreeRTOS, SafeRTOS, Freescale MQX, Nucleus, TinyOS, HP-UX

Funkcjonalność bibliotek kryptograficznych - przykłady

Key generation and exchange [edit]

Implementation	ECDH	DH	DSA	RSA	ElGamal	NTRU	DSS
Botan	Yes	Yes	Yes	Yes	Yes	No	Yes
Bouncy Castle	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Elliptic curve cryptography (ECC) support [edit]

Implementation	NIST	SECG	ECC Brainpool	ECDSA	ECDH	Curve25519	EdDSA	GOST R 34.10
Botan	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bouncy Castle	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
cryptlib	Yes	Yes	Yes	Yes	Yes	No	No	No

Block cipher algorithms [edit]

Implementation	AES	Camellia	3DES	Blowfish	Twofish	CAST5	IDEA	GOST 28147-89 / GOST R 34.12-2015	ARIA
Botan	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bouncy Castle[22]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
cryptlib[23]	Yes	No	Yes	Yes		Yes	Yes		
Crypto++	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes[a]	Yes
Libgcrypt	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
libsodium	Yes[b]	No	No	No	No	No	No	No	No
Nettle	Yes	Yes	Yes	Yes					
OpenSSL	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
wolfCrypt	Yes	Yes	Yes	No	No	No	Yes	No	No

Wsparcie kryptograficznych rozwiązań sprzętowych przez biblioteki kryptograficzne - przykłady

Smartcard, SIM and HSM protocol support [ed

Implementation	PKCS #11	PC/SC	CCID
Botan	Yes	No	No
Bouncy Castle	Yes [a]	No	No
cryptlib	Yes	No	No
Crypto++	No	No	No
Libgcrypt	Yes [26]	Yes [27]	Yes [28]
libsodium	No	No	No
OpenSSL	Yes [29]	No	No
wolfCrypt	Yes	No	No

General purpose CPU / platform acceleration support [edit]

Implementation	AES-NI	SSSE3 / SSE4.1	AVX / AVX2	RdRand	VIA PadLock	Intel QuickAssist	AltiVec[a]
Botan	Yes	Yes	Yes	Yes	No	No	Yes
cryptlib	Yes	Yes	Yes	Yes	Yes	No	No
Crypto++	Yes	Yes	Yes	Yes	Yes[b]	No	Yes
Libgcrypt[31]	Yes	Yes	Yes	Yes	Yes	No	No
libsodium	Yes	Yes	Yes	No	No	No	No
OpenSSL	Yes	Yes	Yes	Yes[c]	Yes	No	Yes
wolfCrypt	Yes	No	Yes	Yes	No	Yes[32]	No

MODUŁ KRYPTOGRAFICZNY

wydzielony moduł sprzętowy (hardware) i/ lub programowy (software, firmware) zawierający zaimplementowane mechanizmy, procesy i/lub algorytmy kryptograficzne i umieszczony w obszarze kryptograficznym (cryptographic boundary)

Zastosowanie modułów kryptograficznych

- *Przechowywanie kluczy kryptograficznych i materiału kluczowego*
- *Szyfrowanie i deszyfrowanie danych*
- *Obliczanie wartości podpisów cyfrowych i ich weryfikacja*
- *Tworzenie logicznej zawartości tzw. tokenów dla protokołów uwierzytelniania*
- *Bezpieczne zarządzanie kluczami i materiałem kluczowym (dystrybucja kluczy, archiwizacja, itp.)*
- *Odtwarzanie sekretów podzielonych z zastosowaniem metod progowych, itp.*

Przykłady sprzętowych modułów kryptograficznych

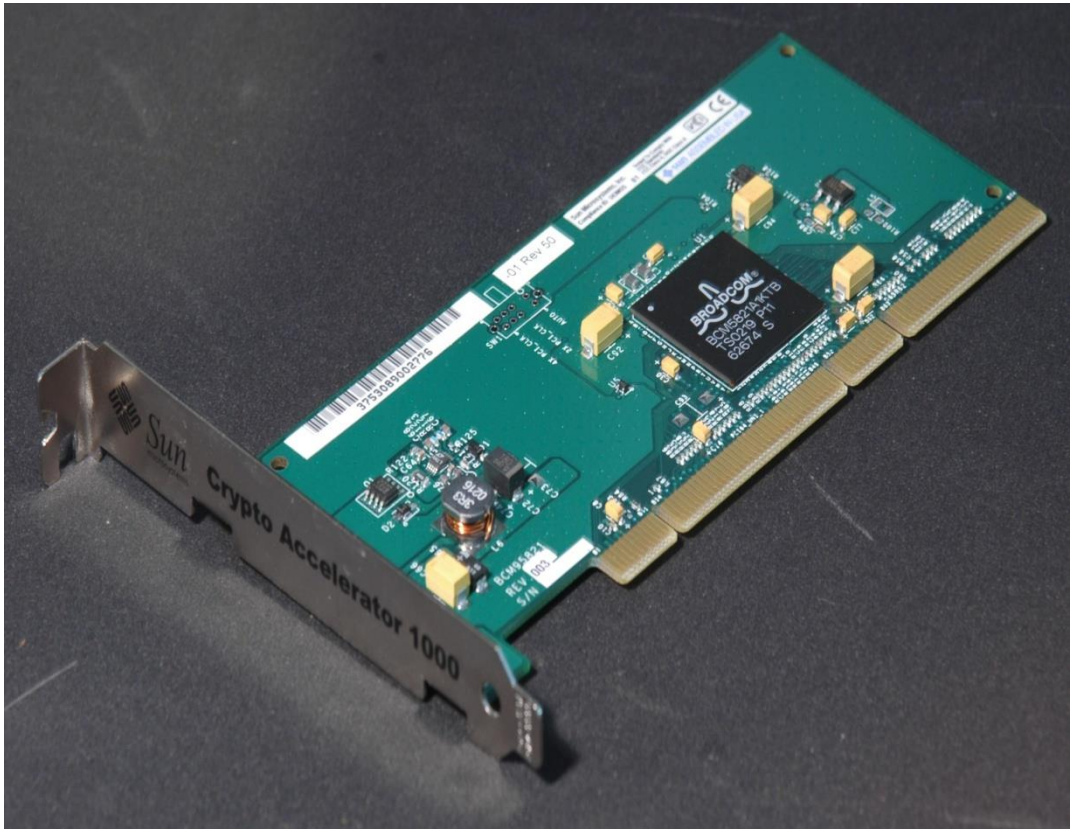
Elektroniczne karty identyfikacyjne (ICCs - Integrated Circuit(s) Cards)

Sprzętowe moduły bezpieczeństwa (HSMs - Hardware Security Modules)

(SAMs – Security Application Modules)

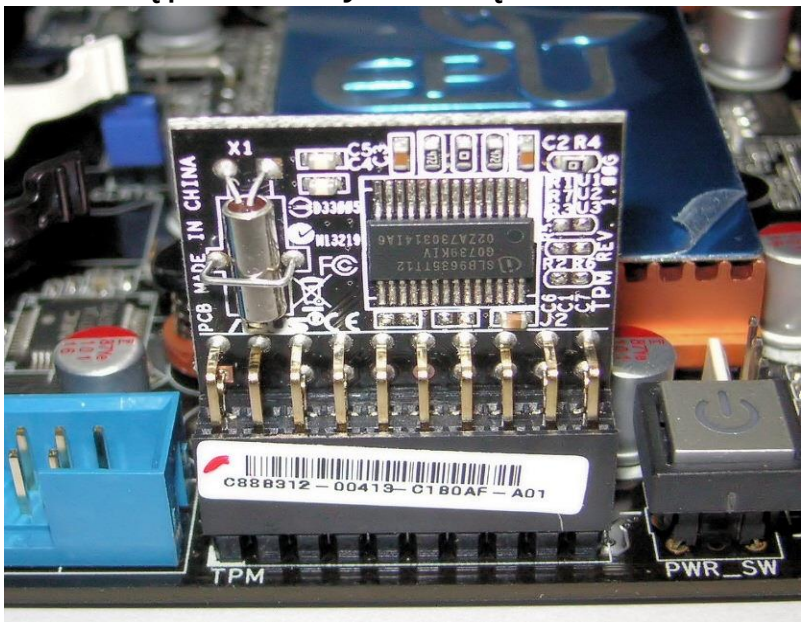
PIN-pady

Przykład akceleratora kryptograficznego SSL/TLS (SUN Crypto Accelerator – 1000)

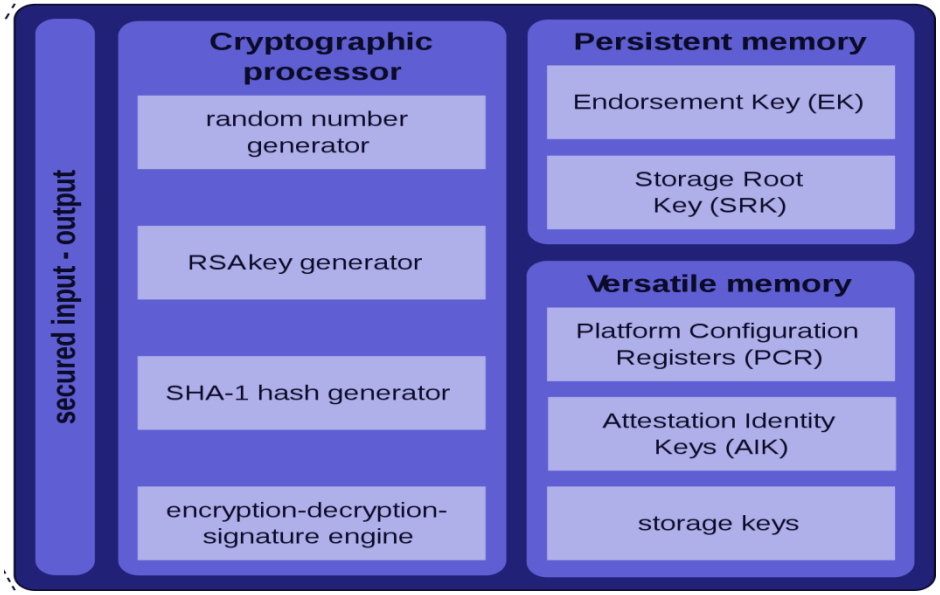


TPM – Trusted Platform Module

Specyfikacja „zaufanego” modułu kryptograficznego firmowana przez Trusted Computing Group (utworzoną przez AMD, HP, Intel, IBM i Microsoft; ostatnia specyfikacja: „Trusted Platform Module Library Specification, Family “2.0”, Level 00, Revision 01.38 – September 2016” (+ poprawki ze stycznia 2018), opublikowana także jako ISO/IEC 11889:2015, części 1-4), zwanego także "TPM chip" lub "TPM Security Device”, będącego częścią PC i umożliwiającego min. generowanie kluczy kryptograficznych i ich bezpieczne użytkowanie i przechowywanie, weryfikację integralności zawartości pamięci (w tym software’u), a także silne uwierzytelnianie urządzenia przy próbach pozyskiwania dostępu do innych urządzeń.



TPM Asus



Is TPM really Trusted ?

2010 – Ch. Tarnovsky przedstawił na Black Hat Briefings atak na TPM polegający na „podsłuchiwaniu” za pomocą sondy wewnętrznej magistrali układu Infineon SLE 66 CL PC. Twierdził, że po 6 miesiącach pracy „wydłubał” z TPM sekrety.

2015 – jako „pokłosie” afery Snowdena ujawniono, że w 2010 zespół CIA twierdził na wewnętrznej konferencji, że przeprowadził atak DPA na moduły TPM wydobywając z nich sekrety.

2017 - biblioteka kodów opracowana przez Infineon, która była szeroko stosowana w modułach TPM dostarczanych przez tą firmę, zawierała lukę (znaną jako ROCA), która pozwalała na wnioskowanie o kluczach prywatnych RSA na podstawie kluczy publicznych.

Dobra praktyka – w aplikacjach kryptograficznych przechowujących klucze szyfrujące bezpośrednio w module TPM należy stosować „maskowanie” tych kluczy (blinding).

PKCS#11

PKCS #11: Cryptographic Token Interface Standard (ostatnia wersja „pod patronatem" RSA Laboratories - v2.20 z *28 czerwca 2004*):

Specyfikacja (standard *de facto*) określająca kryptograficzny interfejs programowy (CAPI) dla urządzeń przechowujących dane kryptograficzne i wykonujących operacje kryptograficzne.

Interfejs nosi nazwę własną CRYPTOKI.

Umożliwia uzyskanie interfejsu niezależnego od urządzenia oraz umożliwiającego współdzielenie zasobów widzianych jako logiczne urządzenie, zwane „tokenem kryptograficznym" (wiele aplikacji komunikujących się z wieloma „urządzeniami" jednocześnie).

Specyfikacja odwołuje się do notacji ASN.1 i specyfikacji ANSI C.

Grudzień 2012 „RSA announce that PKCS #11 management is being transitioned to OASIS (Organization for the Advancement of Structured Information Standards)".

Stan obecny: **OASIS Standard 14 July 2020**

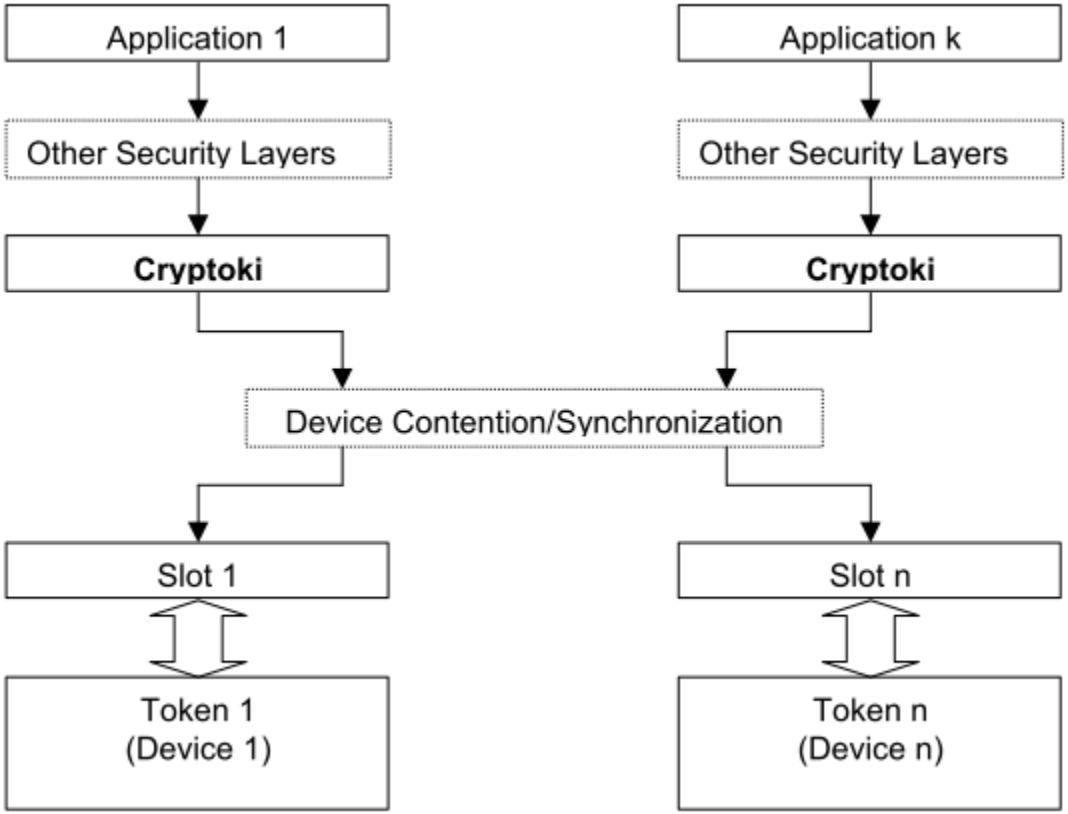
PKCS #11 Cryptographic Token Interface Base Specification Version 3.0

PKCS #11 Cryptographic Token Interface Profiles Version 3.0

PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 3.0

PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification Version 3.0

PKCS#11 – Ogólny model CRYPTOKI



PKCS#11 – tryby R/O i R/W

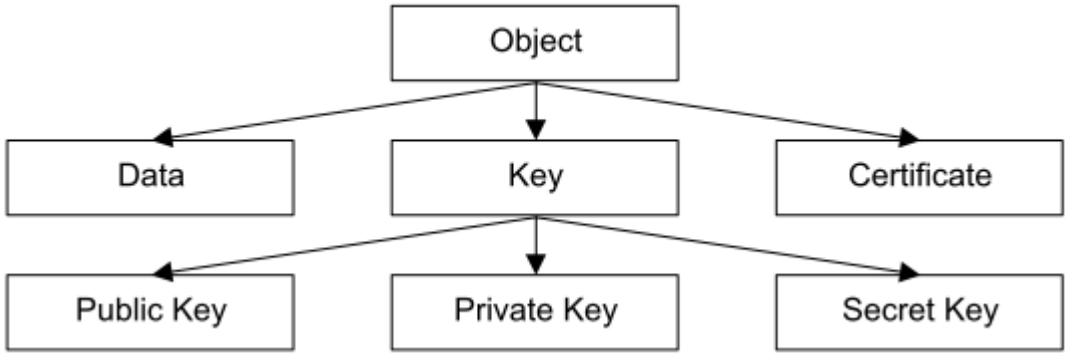
Aplikacja może otworzyć jedną lub wiele sesji komunikacyjnych z tokenem; każda z sesji może być w trybie R/O (Read Only) lub R/W (Read/Write), zaś wybór trybu decyduje o możliwościach manipulacji obiektami w tokenie.

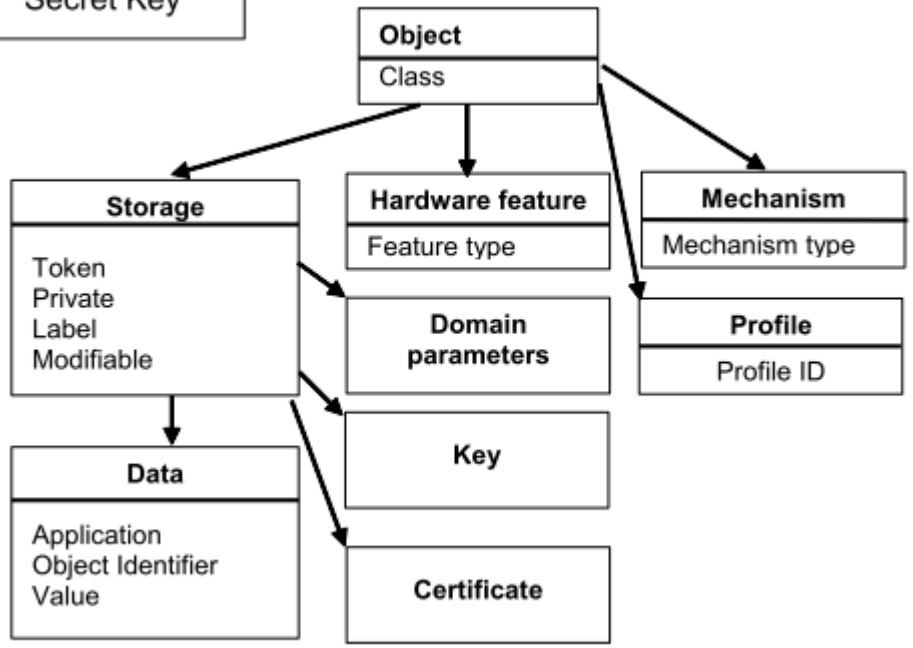
Dopóki użytkownik się nie zaloguje, to otwarta sesja ma charakter „publiczny”.

Dostęp do obiektów różnego typu w zależności od trybu otwarcia sesji

Type of object	Type of session				
	R/O Public	R/W Public	R/O User	R/W User	R/W SO
Public session object	R/W	R/W	R/W	R/W	R/W
Private session object			R/W	R/W	
Public token object	R/O	R/W	R/O	R/W	R/W
Private token object			R/O	R/W	

PKCS#11 – typy obiektów i ich atrybuty





PKCS#11 - Funkcje

Funkcje CRYPTOKI pogrupowane są w następujące kategorie:

- ogólnego przeznaczenia (general-purpose functions - 4)
- zarządzania slotami i tokenami (slot and token management functions - 9)
- zarządzania sesjami (session management functions – 8)
- zarządzania obiektami (object management functions - 9)
- szyfrowania (encryption functions - 4)
- szyfrowania „wielu wiadomości”, w tym AEAD (message-based encryption functions - 5)
- deszyfrowania (decryption functions - 4)
- skracania wiadomości (message digesting functions - 5)
- podpisywania i obliczania MAC (signing and MACing functions – 6)
- weryfikacji podpisów i MAC (functions for verifying signatures and MACs - 6)
- kryptograficzne podwójnego przeznaczenia (dual-purpose cryptographic functions - 4)
- zarządzania kluczami (key management functions - 5)
- generowania liczb losowych (random number generation functions - 2)
- zarządzania funkcjami równoległymi (parallel function management functions - 2)

PKCS#11 – prefiksy i funkcje kryptograficzne

Prefiksy

Prefix	Description
C_	Function
CK_	Data type or general constant
CKA_	Attribute
CKC_	Certificate type
CKD_	Key derivation function
CKF_	Bit flag
CKG_	Mask generation function
CKH_	Hardware feature type
CKK_	Key type
CKM_	Mechanism type
CKN_	Notification
CKO_	Object class
CKP_	Pseudo-random function
CKS_	Session state
CKR_	Return value
CKU_	User type
CKZ_	Salt/Encoding parameter source
h	a handle
ul	a CK_ULONG
p	a pointer
pb	a pointer to a CK_BYTE
ph	a pointer to a handle
pul	a pointer to a CK_ULONG

Funkcje kryptograficzne

Encryption functions	C_EncryptInit	initializes an encryption operation
	C_Encrypt	encrypts single-part data
	C_EncryptUpdate	continues a multiple-part encryption operation
	C_EncryptFinal	finishes a multiple-part encryption operation
Decryption functions	C_DecryptInit	initializes a decryption operation
	C_Decrypt	decrypts single-part encrypted data
	C_DecryptUpdate	continues a multiple-part decryption operation
	C_DecryptFinal	finishes a multiple-part decryption operation
Message digesting functions	C_DigestInit	initializes a message-digesting operation
	C_Digest	digests single-part data
	C_DigestUpdate	continues a multiple-part digesting operation
	C_DigestKey	digests a key
	C_DigestFinal	finishes a multiple-part digesting operation
Dual-purpose cryptographic functions	C_DigestEncryptUpdate	continues simultaneous multiple-part digesting and encryption operations
	C_DecryptDigestUpdate	continues simultaneous multiple-part decryption and digesting operations
	C_SignEncryptUpdate	continues simultaneous multiple-part signature and encryption operations
	C_DecryptVerifyUpdate	continues simultaneous multiple-part decryption and verification operations
Key management functions	C_GenerateKey	generates a secret key
	C_GenerateKeyPair	generates a public-key/private-key pair
	C_WrapKey	wraps (encrypts) a key
	C_UnwrapKey	unwraps (decrypts) a key
	C_DeriveKey	derives a key from a base key
Random number generation functions	C_SeedRandom	mixes in additional seed material to the random number generator
	C_GenerateRandom	generates random data

PKCS#11 – Funkcje kryptograficzne (cd.)

Signing and MACing functions	C_SignInit	initializes a signature operation
	C_Sign	signs single-part data
	C_SignUpdate	continues a multiple-part signature operation
	C_SignFinal	finishes a multiple-part signature operation
	C_SignRecoverInit	initializes a signature operation, where the data can be recovered from the signature
	C_SignRecover	signs single-part data, where the data can be recovered from the signature
Functions for verifying signatures and MACs	C_VerifyInit	initializes a verification operation
	C_Verify	verifies a signature on single-part data
	C_VerifyUpdate	continues a multiple-part verification operation
	C_VerifyFinal	finishes a multiple-part verification operation
	C_VerifyRecoverInit	initializes a verification operation where the data is recovered from the signature
	C_VerifyRecover	verifies a signature on single-part data, where the data is recovered from the signature

PKCS#11 – Mechanizmy

Mechanizm dokładnie określa, w jaki sposób ma być wykonany określony proces kryptograficzny. Ten sposób to wskazanie algorytmu i parametrów definiujących wariant tego algorytmu.

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR [1]	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_RSA_PKCS_KEY_PAIR_GEN					✓		
CKM_RSA_X9_31_KEY_PAIR_GEN					✓		
CKM_RSA_PKCS	✓[2]	✓[2]	✓			✓	
CKM_RSA_PKCS_OAEP	✓[2]					✓	
CKM_RSA_PKCS_PSS		✓[2]					
CKM_RSA_9796		✓[2]	✓				
CKM_RSA_X_509	✓[2]	✓[2]	✓				
CKM_RSA_X9_31		✓[2]					
CKM_SHA1_RSA_PKCS		✓					
CKM_SHA256_RSA_PKCS		✓					
CKM_SHA384_RSA_PKCS		✓					

Standard wskazuje powiązanie mechanizmów z funkcjami.

```
#define CKM_RSA_PKCS_KEY_PAIR_GEN       0x00000000
#define CKM_RSA_PKCS                    0x00000001
#define CKM_RSA_9796                    0x00000002
#define CKM_RSA_X_509                   0x00000003
#define CKM_MD2_RSA_PKCS                0x00000004
#define CKM_MD5_RSA_PKCS                0x00000005
#define CKM_SHA1_RSA_PKCS               0x00000006
#define CKM_RIPEMD128_RSA_PKCS          0x00000007
#define CKM_RIPEMD160_RSA_PKCS          0x00000008
#define CKM_RSA_PKCS_OAEP               0x00000009
#define CKM_RSA_X9_31_KEY_PAIR_GEN      0x0000000A
#define CKM_RSA_X9_31                   0x0000000B
#define CKM_SHA1_RSA_X9_31              0x0000000C
#define CKM_RSA_PKCS_PSS                0x0000000D
#define CKM_SHA1_RSA_PKCS_PSS           0x0000000E
```

Każdy mechanizm ma swój specyficzny unikalny kod.

PKCS#11 – przykład kodu

Przykład szyfrowania za pomocą algorytmu DES w trybie CBC:

```
#define PLAINTEXT_BUF_SZ 200
#define CIPHERTEXT_BUF_SZ 256
CK_ULONG firstPieceLen, secondPieceLen;
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM mechanism = {
CKM_DES_CBC_PAD, iv, sizeof(iv)
};
CK_BYTE data[PLAINTEXT_BUF_SZ];
CK_BYTE encryptedData[CIPHERTEXT_BUF_SZ];
CK_ULONG ulEncryptedData1Len;
CK_ULONG ulEncryptedData2Len;
CK_ULONG ulEncryptedData3Len;
CK_RV rv;
.
.
firstPieceLen = 90;
secondPieceLen = PLAINTEXT_BUF_SZ-firstPieceLen;
rv = C_EncryptInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
        /* Encrypt first piece */
        ulEncryptedData1Len = sizeof(encryptedData);
        rv = C_EncryptUpdate(
         hSession,
         &data[0], firstPieceLen,
         &encryptedData[0], &ulEncryptedData1Len);
if (rv != CKR_OK) {
.
.
}
```

PKCS#11 – przykład kodu (cd.)

Przykład szyfrowania za pomocą algorytmu DES w trybie CBC (cd.):

```
/* Encrypt second piece */
ulEncryptedData2Len = sizeof(encryptedData)-
ulEncryptedData1Len;
rv = C_EncryptUpdate(
        hSession,
        &data[firstPieceLen], secondPieceLen,
        &encryptedData[ulEncryptedData1Len],
          &ulEncryptedData2Len);
if (rv != CKR_OK) {
.
.
}
/* Get last little encrypted bit */
ulEncryptedData3Len =
        sizeof(encryptedData)-ulEncryptedData1LenulEncryptedData2Len;
rv = C_EncryptFinal(
        hSession,
        &encryptedData[ulEncryptedData1Len+ulEncryptedDat
        a2Len],
&ulEncryptedData3Len);
if (rv != CKR_OK) {
.
.
}
}
```

CryptoAPI

Biblioteka CryptoAPI jest interfejsem programowania aplikacji udostępnianym jako część systemu Microsoft Windows.

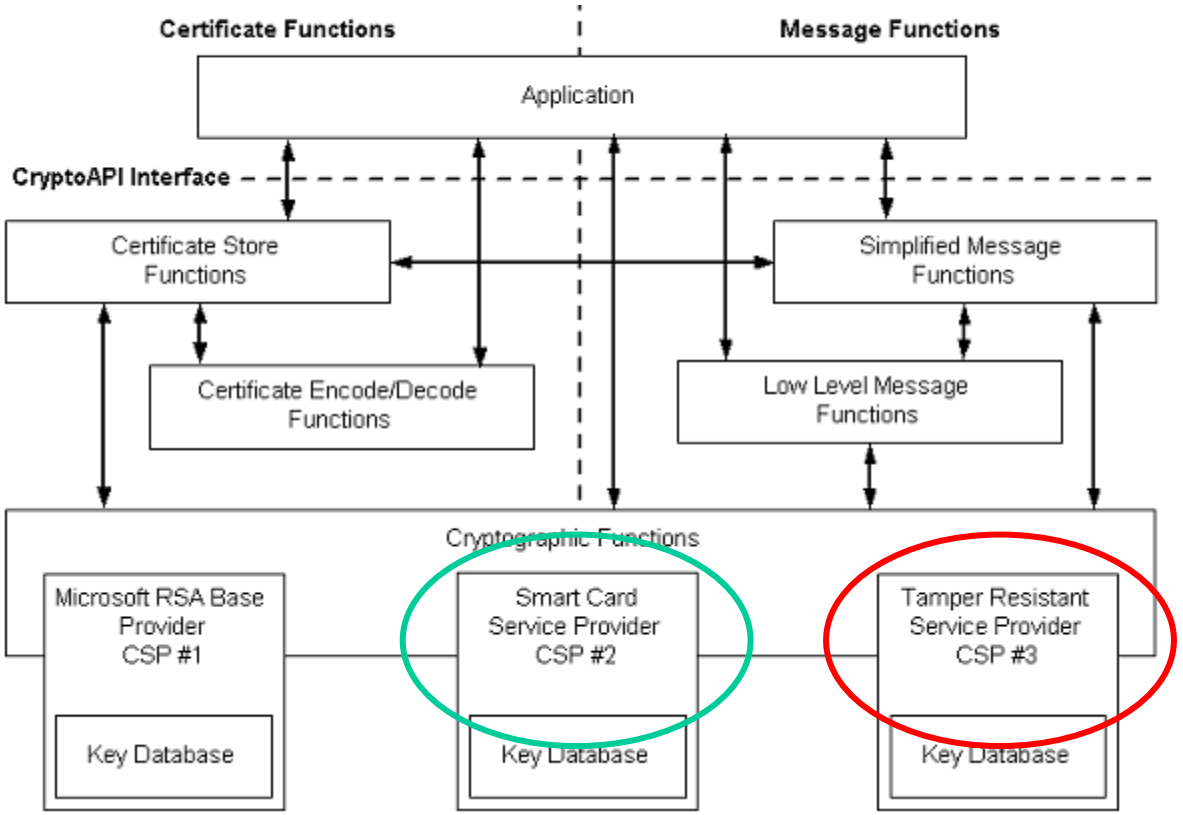
Interfejs CryptoAPI zawiera zestaw funkcji, które pozwalają aplikacjom wykorzystywać metody i mechanizmy kryptograficzne.

Systemy z rodziny Windows były wyposażone w pakiet CryptoAPI począwszy od wersji 95 oraz NT 4.0 z zainstalowanym produktem (przeglądarką internetową) Internet Explorer w wersji 4.0.

Wszystkie operacje wykonywane są za pośrednictwem dostawców usług kryptograficznych (Cryptographic Service Providers - CSP). Są to niezależne moduły, zawierające gotowe implementacje algorytmów kryptograficznych, a także zarządzające kluczami użytkownika. Istnieje możliwość rozbudowania systemu o dodatkowe moduły nowych dostawców.

Następcą CryptoAPI jest Cryptography API Next Generation (CNG) .

CryptoAPI - architektura



Źródło: Platform SDK: Cryptography, „CryptoAPI System Architecture”, http://msdn.microsoft.com/library/en-us/seccrypto/security/cryptoapi_system_architecture.asp

CryptoAPI – architektura (cd.)

(Podstawowe) funkcje kryptograficzne (*ang. (base) cryptographic functions*) :

- ✓ **funkcje kontekstowe wykorzystywane do połączenia z CSP; te funkcje umożliwiają aplikacjom wybór określonego CSP przez nazwę lub wybór określonego CSP realizującego potrzebną klasę funkcjonalności;**
- ✓ **funkcje generowania kluczy wykorzystywane do generowania i przechowywania kluczy kryptograficznych; uwzględniają różne tryby szyfrowania, wektory inicjujące i inne cechy mechanizmów kryptograficznych;**
- ✓ **funkcje wymiany kluczy wykorzystywane do bezpiecznego uzgadniania lub przesyłania kluczy.**

Funkcje szyfrowania/deszyfrowania (*ang. (certificate) encode/decode functions*):
służące do szyfrowania/deszyfrowania danych, a także obliczania wartości funkcji skrótu.

Funkcje magazynu certyfikatów (*ang. certificate store functions*):
wykorzystywane do zarządzania zbiorami cyfrowych certyfikatów, umożliwiają min. sprawdzanie certyfikatów kluczy publicznych zawartych w dokumentach.

Uproszczone funkcje wiadomości (*ang. simplified message functions*):
funkcje wysokiego poziomu wykorzystywane do szyfrowania i deszyfrowania wiadomości i danych, podpisywania wiadomości i danych oraz weryfikowania autentyczności podpisów na odbieranych wiadomościach i danych.

Niskopoziomowe funkcje wiadomości (*ang. low level message functions*):
wykorzystywane podczas realizacji wszystkich zadań wykonywanych przez uproszczone funkcje wiadomości, umożliwiają pełniejszą kontrolę nad wykonywanymi operacjami, ale wymagają większej liczby wywołań funkcji.

CryptoAPI – typy dostawców usług kryptograficznych

Typy dostawców CSP różnią się funkcjonalnością ...

- ✓ PROV_RSA_FULL
- ✓ PROV_RSA_AES
- ✓ PROV_RSA_SIG
- ✓ PROV_RSA_SCHANNEL
- ✓ PROV_DSS
- ✓ PROV_DSS_DH
- ✓ PROV_DH_SCHANNEL
- ✓ PROV_FORTEZZA
- ✓ PROV_MS_EXCHANGE
- ✓ PROV_SSL

Key exchange:	RSA
Signature:	RSA
Encryption:	RC2, RC4
Hashing:	MD5, SHA-1

Key exchange:	None
Signature:	RSA
Encryption:	None
Hashing:	MD5, SHA-1

Key exchange:	KEA
Signature:	DSS
Encryption:	Skipjack
Hashing:	SHA-1

CryptoAPI – typy dostawców usług kryptograficznych (c.d.)

... i siłą mechanizmów kryptograficznych

- ✓ Base Cryptographic Provider
- ✓ Strong Cryptographic Provider
- ✓ Enhanced Cryptographic Provider

RSA signature:	512 bits
RSA key exchange:	512 bits
RC2 block cipher:	40 bits
RC4 stream cipher:	40 bits
DES:	56 bits
3DES (2 keys):	not supported
3DES (3 keys):	not supported

RSA signature:	1024 bits

Key length: Can be set, 384 bits to 16,384 bits in 8-bit increments. Default key length: 1,024 bits.

RSA key exchange:	1024 bits
RC2 block cipher:	128 bits
RC4 stream cipher:	128 bits
DES:	56 bits
3DES (2 keys):	112 bits
3DES (3 keys):	168 bits

For RC2 & RC4 salt length can be set

RSA signature:	1024 bits
RSA key exchange:	1024 bits
RC2 block cipher:	128 bits
RC4 stream cipher:	128 bits
DES:	56 bits
3DES (2 keys):	112 bits
3DES (3 keys):	168 bits

CryptoAPI – typy dostawców usług kryptograficznych (c.d.)

Microsoft AES Cryptographic Provider	Microsoft Enhanced Cryptographic Provider with support for AES encryption algorithms.
Microsoft DSS Cryptographic Provider	Provides hashing, data signing, and signature verification capability using the Secure Hash Algorithm (*SHA*) and Digital Signature Standard (DSS) algorithms.
Microsoft Base DSS and Diffie-Hellman Cryptographic Provider	A superset of the DSS Cryptographic Provider that also supports Diffie-Hellman key exchange, hashing, data signing, and signature verification using the Secure Hash Algorithm (SHA) and Digital Signature Standard (DSS) algorithms.
Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider	Supports Diffie-Hellman key exchange (a 40-bit DES derivative), SHA hashing, DSS data signing, and DSS signature verification.
Microsoft DSS and Diffie-Hellman/Schannel Cryptographic Provider	Supports hashing, data signing with DSS, generating Diffie-Hellman (D-H) keys, exchanging D-H keys, and exporting a D-H key. This CSP supports key derivation for the SSL3 and TLS1 protocols.
Microsoft RSA/Schannel Cryptographic Provider	Supports hashing, data signing, and signature verification. The algorithm identifier CALG_SSL3_SHAMD5 is used for SSL 3.0 and TLS 1.0 client authentication. This CSP supports key derivation for the SSL2, PCT1, SSL3 and TLS1 protocols.

CryptoAPI – przykłady wywołania funkcji informujących o dostępnych typach i sile CSP

Funkcja `CryptEnumProvidersTypes`

Przykład wywołania:

```
c:\Documents and Settings\nr0\Moje dokumenty\Visual Studio Projects\enum_prov_types\Debug\e...
Dostepne typy dostawcow uslug kryptograficznych:
Typ dostawcy    Nazwa typu dostawcy
------------    -------------------------------------------
           1    RSA Full (Signature and Key Exchange)
           3    DSS Signature
          12    RSA SChannel
          13    DSS Signature with Diffie-Hellman Key Exchange
          18    Diffie-Hellman SChannel
          24    RSA Full and AES
```

Funkcja `CryptEnumProviders`

Przykład wywołania:

CSP dla ICC

```
c:\documents and settings\nr0\moje dokumenty\visual studio projects\enum_prov\debug\enum_prov.exe
Lista dostepnych dostawcow uslug kryptograficznych:
Typ dostawcy    Nazwa dostawcy
------------    -------------------------------------------
           1    Gemplus GemSAFE Card CSP v1.0
           1    Infineon SICRYPT Base Smart Card CSP
           1    Microsoft Base Cryptographic Provider v1.0
          13    Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
           3    Microsoft Base DSS Cryptographic Provider
          18    Microsoft DH SChannel Cryptographic Provider
           1    Microsoft Enhanced Cryptographic Provider v1.0
          13    Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
          24    Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
          12    Microsoft RSA SChannel Cryptographic Provider
           1    Microsoft Strong Cryptographic Provider
           1    Schlumberger Cryptographic Service Provider
```

CryptoAPI – przykład kodu

Przykład wywołania sekwencji realizacji podpisu cyfrowego:

```
//--------------------------------------------------------------------
// Copyright (c) Microsoft Corporation.  All rights reserved.
#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>
#include <conio.h>
#define MY_ENCODING_TYPE  (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
void MyHandleError(char *s);
void Get_And_Print_Hash(HCRYPTHASH hHash);

void main(void)
{
//--------------------------------------------------------------------
HCRYPTPROV hProv;
BYTE *pbBuffer= (BYTE *)„Data to be signed";
DWORD dwBufferLen = strlen((char *)pbBuffer)+1;
HCRYPTHASH hHash;
HCRYPTKEY hKey;
HCRYPTKEY hPubKey;
BYTE *pbSignature;
DWORD dwSigLen,a;
DWORD dwBlobLen;
LPTSTR szDescription = "Test Data Description";
//--------------------------------------------------------------------
if(CryptAcquireContext(
   &hProv,
   NULL,
   NULL,
   PROV_RSA_FULL,//PROV_RSA_FULL,
   0))
{
}
else
{
    MyHandleError("Error during CryptAcquireContext.");
}
```

CryptoAPI – przykład kodu (cd.)

Przykład wywołania sekwencji realizacji podpisu cyfrowego (cd.):

```
//--------------------------------------------------------------------
if(CryptGetUserKey(
   hProv,
   AT_SIGNATURE,
   &hKey))
{
    printf(„Access to signing key obtained. \n");
}
else
{
    MyHandleError("Error during CryptGetUserKey for signing key.");
}
//--------------------------------------------------------------------
DWORD dwCount;
BYTE pbData[16];
if(CryptGetKeyParam(hKey,KP_KEYLEN,pbData,&dwCount,0))
{printf("\nKey length : %d bits\n",*pbData);}
else
{
MyHandleError("Error getting the KP_KEYLEN.");
}
//--------------------------------------------------------------------
if(CryptExportKey(
   hKey,
   NULL,
   PUBLICKEYBLOB,
   0,
   pbKeyBlob,
   &dwBlobLen))
{
printf("\nPublic key exported. \n");
        printf(„Length of PUBLICKEYBLOB : %d bits\n",dwBlobLen);
}
else
{
    MyHandleError("Error during CryptExportKey.");
}
```

CryptoAPI – przykład kodu (cd.)

Przykład wywołania sekwencji realizacji podpisu cyfrowego (cd.):

```
//--------------------------------------------------------------------
// Create the hash object.
if(CryptCreateHash(
   hProv,
   CALG_MD5,
   0,
   0,
   &hHash))
{
}
else
{    MyHandleError("Error during CryptCreateHash.");
}
//--------------------------------------------------------------------
// Compute the cryptographic hash of the buffer.
if(CryptHashData(
   hHash,
   pbBuffer,
   dwBufferLen,
   0))
{    printf("\nHash value created.\n");}
else
{    MyHandleError("Error during CryptHashData.");}
//--------------------------------------------------------------------
dwSigLen= 0;
if(CryptSignHash(
   hHash,
   AT_SIGNATURE,
   szDescription,
   CRYPT_NOHASHOID,
   pbSignature,
&dwSigLen))
{    printf(„Signature created.\n");}
else
{    MyHandleError("Error during CryptSignHash.");
}
```

Java Security

Technologia **Java Security** obejmuje obszerny zestaw API, narzędzi i implementacji powszechnie stosowanych algorytmów, mechanizmów i protokołów zapewniających bezpieczeństwo informacji.

Zakres **Java Security API** obejmuje min.:

- kryptografię;
- infrastrukturę klucza publicznego;
- bezpieczną komunikację;
- kontrolę dostępu

Min. w specyfikacji określono API wspierające różne usługi kryptograficzne, takie jak: **podpisy cyfrowe**, **skróty wiadomości**, **szyfry (symetryczne i asymetryczne, blokowe i strumieniowe)**, **kody uwierzytelniania wiadomości (MAC)**, **generatory kluczy kryptograficznych**, itp.

Uwzględniono także wsparcie dla tokenów kryptograficznych zgodnych z **PKCS#11**.

Java Security

W celu pozyskania dostępu do określonej usługi związanej z bezpieczeństwem (algorytmu, mechanizmu, protokołu) aplikacja odwołuje się do odpowiedniej metody `getInstance`.

Np. obliczenie wartości funkcji skrótu wymaga wywołania metody z klasy `java.security.MessageDigest`.

Przykład wywołania funkcji skrótu SHA-256:

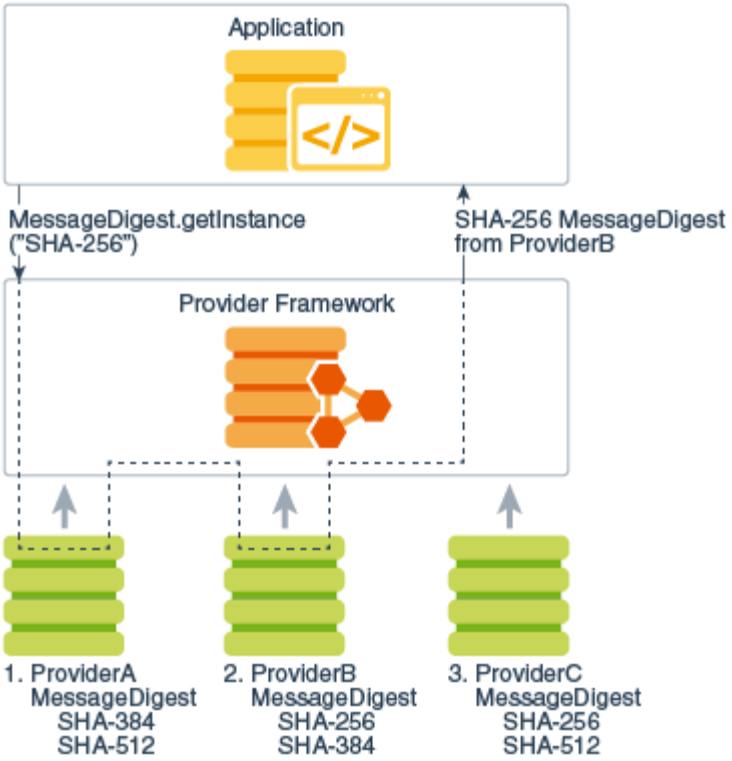
```
MessageDigest md = MessageDigest.getInstance(„SHA-256");
```

Program może opcjonalnie żądać implementacji od określonego dostawcy usług kryptograficznych przez wskazanie jego nazwy, np.:

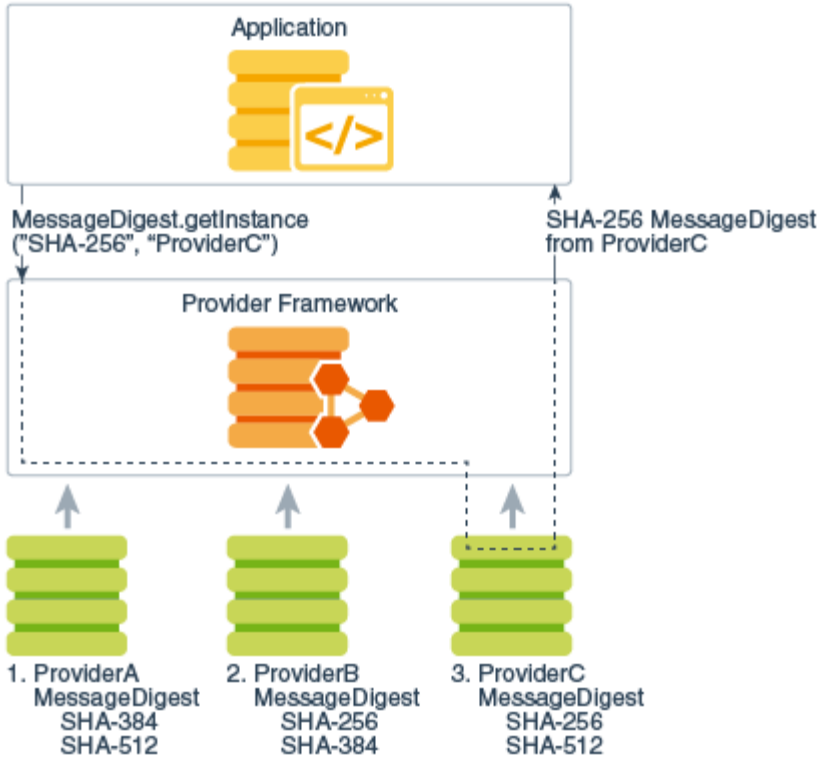
```
MessageDigest md = MessageDigest.getInstance(„SHA-256", "ProviderC");
```

Java – wybór dostawcy usługi kryptograficznej

Dwa różne mechanizmy wywołania tej samej usługi:



Wyszukanie dostawcy



Wybór wskazanego dostawcy

Źródło: https://docs.oracle.com/javase/9/security/java-security-overview1.htm#JSSEC-GUID-0C458D46-BA4F-4091-817B-9902B6E18240

Java - Pakiety i klasy Javy związane z bezpieczeństwem, w tym z mechanizmami kryptograficznymi - przykłady

Package	Class/Interface Name	Usage	Module
java.lang	SecurityException	Indicates a security violation	java.base
java.lang	SecurityManager	Mediates all access control decisions	java.base
java.lang	System	Installs the SecurityManager	java.base
java.security	AccessController	Called by default implementation of SecurityManager to make access control decisions	java.base
java.security	DomainLoadStoreParameter	Stores parameters for the Domain keystore (DKS)	java.base
java.security	Key	Represents a cryptographic key	java.base
java.security	KeyStore	Represents a repository of keys and trusted certificates	java.base
java.security	MessageDigest	Represents a message digest	java.base
java.security	Permission	Represents access to a particular resource	java.base
java.security	PKCS12Attribute	Supports attributes in PKCS12 keystores	java.base
java.security	Policy	Encapsulates the security policy	java.base

Zródło: https://docs.oracle.com/javase/9/security/java-security-overview1.htm#JSSEC-GUID-CF323502-F719-4618-91FE-4D37CB57FF24

Java – przykład kodu

Przykład wywołania sekwencji realizacji podpisu cyfrowego za pomocą algorytmu DSA i jego weryfikacji:

```
//-------------------------------------------------------------------
import java.io.FileOutputStream;
import java.security.*;
import javax.crypto.*;
import java.util.*;

public class MainClass {
        public static void main(String[] args) {
                String msg = „Tekst probny";
                String alg = "DSA";
                KeyPair keys;
                KeyPairGenerator gen;
                Signature sig;
                java.io.FileOutputStream file;
                byte[] signature;
                try {
                                file = new FileOutputStream("keys.txt");
                                gen = KeyPairGenerator.getInstance(alg);
                                gen.initialize(1024, new SecureRandom());
                                keys = gen.generateKeyPair();
                                file.write(keys.getPublic().getEncoded());
                                sig = Signature.getInstance(alg);
                                sig.initSign(keys.getPrivate());
                                sig.update(msg.getBytes());
                                signature = sig.sign();
                                sig.initVerify(keys.getPublic());
                                sig.update(msg.getBytes());
                                if(sig.verify(signature))
                                {System.out.println("Poprawny");}
                                else
                                {System.out.println("Niepoprawny");            }

                } catch (Exception e) {
                                e.printStackTrace();
                }
        }}
```

OpenSSL

Biblioteka OpenSSL wyewoluowała z utworzonej przez Erica Younga biblioteki SSLeay, bezpłatnej implementacji protokołu SSL (Secure Socket Layer). Zasadniczym jej przeznaczeniem jest obsługa protokołu SSL opublikowanego przez firmę Netscape, oraz jego późniejszej modyfikacji – TLS (Transport Layer Security).

Oba protokoły służą głównie do utworzenia bezpiecznego połączenia między dwoma węzłami sieci komputerowej.

Szczegółowe informacje: http://www.openssl.org

OpenSSL

Struktura opisująca algorytm szyfrujący:

```
typedef struct ssl_cipher_st
    {
    int valid;
    const char *name;                   /* text name */
    unsigned long id;                   /* id, 4 bytes, first is version */
    unsigned long algorithms;           /* what ciphers are used */
    unsigned long algo_strength;        /* strength and export flags */
    unsigned long algorithm2;           /* Extra flags */
    int strength_bits;                  /* Number of bits really used */
    int alg_bits;                       /* Number of bits for algorithm */
    unsigned long mask;                 /* used for matching */
    unsigned long mask_strength;        /* also used for matching */
    } SSL_CIPHER;
```

OpenSSL

Przykład kodu - procedura szyfrowania kluczem prywatnym RSA:

```
int SSL_CTX_use_RSAPrivateKey_file(SSL_CTX *ctx, const char *file, int type)
        {
        int j,ret=0;
        BIO *in;
        RSA *rsa=NULL;
        in=BIO_new(BIO_s_file_internal());
        if (in == NULL)
                {
                SSLerr(SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_FILE,ERR_R_BUF_LIB);   goto end;
                }
        if (BIO_read_filename(in,file) <= 0)
                {
                SSLerr(SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_FILE,ERR_R_SYS_LIB);   goto end;
                }
        if      (type == SSL_FILETYPE_ASN1)
                {
                j=ERR_R_ASN1_LIB;
                rsa=d2i_RSAPrivateKey_bio(in,NULL);
                }
        else if (type == SSL_FILETYPE_PEM)
                {
                j=ERR_R_PEM_LIB;
                rsa=PEM_read_bio_RSAPrivateKey(in,NULL,
                                ctx->default_passwd_callback,ctx->default_passwd_callback_userdata);      }
        else
                {
                SSLerr(SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_FILE,SSL_R_BAD_SSL_FILETYPE);  goto end;
                }
        if (rsa == NULL)
                {
                SSLerr(SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_FILE,j);  goto end;
                }
        ret=SSL_CTX_use_RSAPrivateKey(ctx,rsa);
        RSA_free(rsa);
end:
        if (in != NULL) BIO_free(in);
        return(ret);
        }
```

Koniec części 2