Natalia ŁOMNICKA[1], Karol JABŁOŃSKI[2]

Opiekun naukowy: Jacek LACH[3]

IMPLEMENTACJA ALGORYTMU RSA W ŚRODOWISKU MIKROKONTROLERA

Streszczenie: Zaprezentowano implementację algorytmu RSA w środowisku mikrokontrolera rodziny STM32F1. Program zapisany w pamięci Flash jest wykonywany bezpośrednio na warstwie sprzętowej, co ogranicza obciążenie zasobów. Sterowanie funkcjonalnością odbywa się przy wykorzystaniu interfejsu USB. Przy generowaniu liczb losowych wykorzystano wejście konwertera analogowo-cyfrowego układu. Przeprowadzono również proces emulacji pamięci EEPROM na pamięci Flash.

Słowa kluczowe: kryptografia, kryptografia z kluczem publicznym, szyfrowanie, podpis cyfrowy, mikrokontroler, STM

IMPLEMENTATION OF RSA ALGORITHM FOR MICROCONTROLLER

Summary: Implementation of RSA algorithm for microcontroller of STM32F1 family is presented. The program is written to Flash memory and then executed directly on hardware layer, which decreases the load of the resources. The device is controlled via USB interface. An input of an analog-digital converter was used to generate random numbers. The process of an emulation of EEPROM memory on Flash memory was also conducted.

Keywords: cryptography, public key cryptography, encryption, digital signature, microcontroller, STM

1. Wstęp

Algorytm szyfrowania asymetrycznego RSA został wynaleziony w 1977 roku przez R. Rivesta, A. Shamira i L. Adlemana z Massachusetts Institute of Technology. System RSA po dziś dzień jest najpowszechniej wykorzystywanym spośród

[1] inż., Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki, natalom781@student.polsl.pl
[2] mgr inż., Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Automatyki, karol.jablonski@polsl.pl
[3] dr inż., Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki, jacek.lach@polsl.pl

systemów kryptografii z kluczami publicznymi [1]. Potrzeba rozwiązania zapewniającego pełne utajnienie komunikacji, jak również możliwość pewnego ustalenia nadawcy komunikatu, znajduje zastosowanie w wojskowości, celach komercyjnych i prywatnych. Wykonywanie operacji kryptograficznych oraz przechowywanie danych z nimi związanych na komputerze osobistym powoduje zagrożenie dla ochrony prywatności [2]. Jednym z rozwiązań jest powierzenie tych zadań zewnętrznemu układowi.

Celem pracy jest implementacja algorytmu RSA na zewnętrznym urządzeniu, w środowisku mikrokontrolera. Wybrany mikrokontroler należy do rodziny STM32F1, a jego budowa opiera się na rdzeniu Cortex-M3. Układ nie posiada systemu operacyjnego i wykonuje program zapisany w jego pamięci Flash bezpośrednio na warstwie sprzętowej, co znacznie ogranicza obciążenie zasobów. Sterowanie funkcjonalnością zaprogramowanego układu odbywa się za pomocą komputera osobistego, przy wykorzystaniu interfejsu USB oraz dedykowanej aplikacji wykonanej na potrzeby projektu. Do zaprezentowania działania algorytmu wybrano mechanizm podpisu cyfrowego; przy generowaniu liczb losowych wykorzystano wejście konwertera analogowo-cyfrowego układu. Przeprowadzono również proces emulacji pamięci EEPROM na pamięci Flash.

2. Algorytm RSA

Do połowy XX wieku systemy kryptograficzne opierały się na operacjach podstawienia i permutacji [1]. Do szyfrowania i deszyfracji wykorzystywano ten sam klucz, było to więc szyfrowanie symetryczne. Powodowało to problemy z dystrybucją klucza - klucz należało dostarczyć komunikującym się stronom (co stwarzało niebezpieczeństwo przechwycenia go np. podczas przesyłu w niezabezpieczonej sieci) lub korzystać z pośrednictwa centrum dystrybuującego. Tym samym całkowite utajnienie komunikatów było utrudnione [3].

Kryptosystem z kluczem publicznym wprowadzał dwa różne klucze, dlatego też zwany jest również szyfrowaniem asymetrycznym. Został zaproponowany w 1976 roku przez Diffiego i Hellmana [4]. Za pomocą jednego z kluczy oraz algorytmu szyfrującego tekst jawny przekształcany jest przez nadawcę na szyfrogram i wysyłany odbiorcy. Odbiorca przekształca szyfrogram do tekstu jawnego za pomocą drugiego klucza oraz algorytmu deszyfrującego. Na tej zasadzie opierają się wszystkie algorytmy asymetryczne. Ich podstawową właściwością powinna być duża trudność obliczeniowa odtworzenia klucza deszyfrującego na podstawie znajomości algorytmu i klucza szyfrującego mimo tego, że oba klucze są ze sobą powiązane [1].

Metoda zaproponowana przez Diffiego i Hellmana rozwiązywała również inny problem występujący w przypadku szyfrowania symetrycznego – brak możliwości pewnego ustalenia tożsamości nadawcy komunikatu [3]. Szyfrowanie asymetryczne umożliwia uwierzytelnianie, gdy nadawca wysyła komunikat podpisany własnym kluczem prywatnym, a odbiorca weryfikuje go za pomocą klucza publicznego nadawcy. Jeśli weryfikacja się powiodła, odbiorca ma pewność co do autorstwa wiadomości, ponieważ nikt poza nadawcą nie zna klucza prywatnego nadawcy.

Dodatkowo zapewniana jest również integralność komunikatu – bez znajomości klucza prywatnego nadawcy nie jest możliwa żadna modyfikacja.

Diffie oraz Hellman w swojej pracy [4] sformułowali ogólne warunki, jakie musiałby spełniać bezpieczny asymetryczny algorytm kryptograficzny. Były nimi:

- łatwość generowania pary kluczy,
- łatwość obliczeniowa szyfrowania wiadomości za pomocą klucza publicznego odbiorcy,
- łatwość obliczeniowa odszyfrowania szyfrogramu za pomocą klucza prywatnego odbiorcy,
- nieopłacalność obliczeniowa odtworzenia klucza prywatnego odbiorcy na podstawie jego klucza publicznego,
- nieopłacalność obliczeniowa odtworzenia wiadomości na podstawie znanego szyfrogramu oraz klucza publicznego odbiorcy.

Jednym z pierwszych algorytmów asymetrycznych spełniających te wymagania jest algorytm RSA [5].

Szyfrowanie oraz deszyfrację w algorytmie RSA przedstawiają operacje (1) i (2), gdzie M oznacza szyfrowaną wiadomość, C – szyfrogram, n – maksymalną wartość liczby w bloku. Para $\{e, n\}$ jest kluczem publicznym nadawcy, a para $\{d, n\}$ jego kluczem prywatnym. Wartości n, e znane są nadawcy i adresatowi, natomiast d – jedynie nadawcy.

$$C = M^e \bmod n \tag{1}$$

$$M = C^d \bmod n = \left(M^e\right)^d \bmod n = M^{ed} \bmod n \tag{2}$$

Poniższy opis przedstawia przebieg algorytmu RSA dla podpisu cyfrowego i został opracowany na podstawie [1].

Nadawca generuje parę kluczy: klucz publiczny i klucz prywatny. W tym celu wybiera parę różnych liczb pierwszych p i q. Wyznacza wartości n oraz $\varphi(n)$ według wzorów (3) oraz (4).

$$n = p \cdot q \tag{3}$$

$$\varphi(n) = (p-1)(q-1) \tag{4}$$

Następnie wybiera e spełniające warunki (5), (6).

$$1 < e < \varphi(n) \tag{5}$$

$$NWD(e, \varphi(n)) = 1 \tag{6}$$

Oblicza d według wzoru (7).

$$d \equiv e^{-1} (\bmod \varphi(n)) d \tag{7}$$

Uzyskane klucze publiczny PU i prywatny PR określone są poprzez (8), (9).

$$PU = \{e, n\} \tag{8}$$

$$PR = \{d, n\} \tag{9}$$

Następnie nadawca podpisuje komunikat swoim kluczem prywatnym. Tekst jawny jest przetwarzany blokami, z których każdy jest traktowany jako liczba całkowita mniejsza od n. W rezultacie otrzymuje podpisaną wiadomość S jak we wzorze (10).

$$S = M^d \bmod n \tag{10}$$

Odbiorca po otrzymaniu wiadomości może ją zweryfikować za pomocą klucza publicznego nadawcy jak we wzorze (11).

$$M = S^e \bmod n \tag{11}$$

Podpisywanie całego komunikatu jest jednak mało efektywnym sposobem uzyskania podpisu cyfrowego [1]. Zamiast tego, do zaszyfrowania kluczem prywatnym można wykorzystać autentykator - niewielki blok bitowy tworzony na podstawie wiadomości za pomocą funkcji haszującej. Jakakolwiek zmiana treści wiadomości bez wpływu na autentykator nie powinna być możliwa.

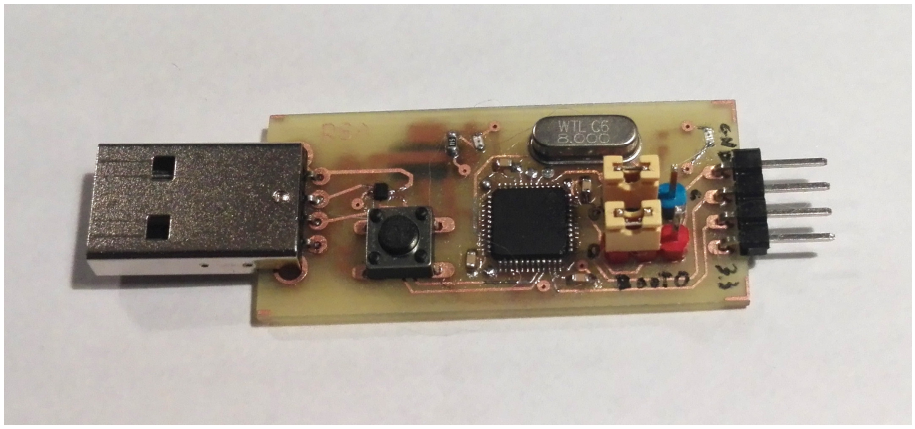
3. Układ uruchomieniowy

Z powodu dużej złożoności obliczeniowej operacji, które musi wykonywać mikrokontroler, konieczne było wybranie takiego układu, który umożliwiałby ich przeprowadzenie w akceptowalnym czasie. Zastosowany 32-bitowy mikrokontroler STM32F103C8T6 pracuje z maksymalną częstotliwością 72MHz oraz posiada 64KB pamięci Flash, obsługuje interfejs USB 2.0, a przy tym jest stosunkowo tani. Ponieważ złącza interfejsu USB w komputerach PC zapewniają napięcie 5 V, a mikrokontroler wymaga napięcia 3.3 V, niezbędne było wykorzystanie stabilizatora (układ LM1117). Użyto również układu NUF2030XV6, który zawiera wymagane przez specyfikację USB rezystory terminujące [7] oraz zapewnia ochronę przed ładunkami elektrostatycznymi.

Powodem wybrania interfejsu USB do komunikacji urządzenia z komputerem PC była przede wszystkim jego powszechność. Interfejs USB stał się podstawowym łączem szeregowym komputerów osobistych i przenośnych, zastępując wiele tradycyjnych portów. Został zaprojektowany w taki sposób, by użytkownik komputera mógł w prosty sposób z niego korzystać – wykrycie urządzenia, jego konfiguracja, a zazwyczaj również instalacja sterownika odbywają się automatycznie. Jest to jeden z niewielu sprawnie działających systemów plug and play [6].

Podczas rozmieszczania elementów przy tworzeniu projektu obwodu drukowanego PCB kierowano się podstawowymi zasadami związanymi z projektowaniem układów elektronicznych: kondensatory zostały umieszczone blisko wyprowadzeń zasilania układów, kwarc blisko mikrokontrolera, układ NUF blisko wtyku USB. Dla wygody użytkownika wtyk USB został umieszczony przy krawędzi płytki, a elementy rozmieszczono maksymalnie ciasno, tak by urządzenie miało wymiary i kształty zbliżone do klasycznej pamięci USB. Po wstępnym rozmieszczeniu elementów ręcznie poprowadzono ścieżki pomiędzy elementami, dbając, by były one możliwie najkrótsze, nie leżały zbyt blisko siebie oraz by ograniczyć liczbę przelotek.

W przypadku linii różnicowej (D+, D-) parę ścieżek umieszczono równolegle, co zmniejsza podatność transmisji na zakłócenia. Projekt płytki składa się z dwóch warstw. Płytka została wykonana na laminacie FR-4, ma rozmiar 21 x 46 mm, a grubość ścieżek miedzi wynosi 18 μm. Rysunek 1 przedstawia górną warstwę gotowego urządzenia.



Rysunek 1. Fotografia gotowego urządzenia USB.

4. Implementacja części programowej

4.1. Wymagania

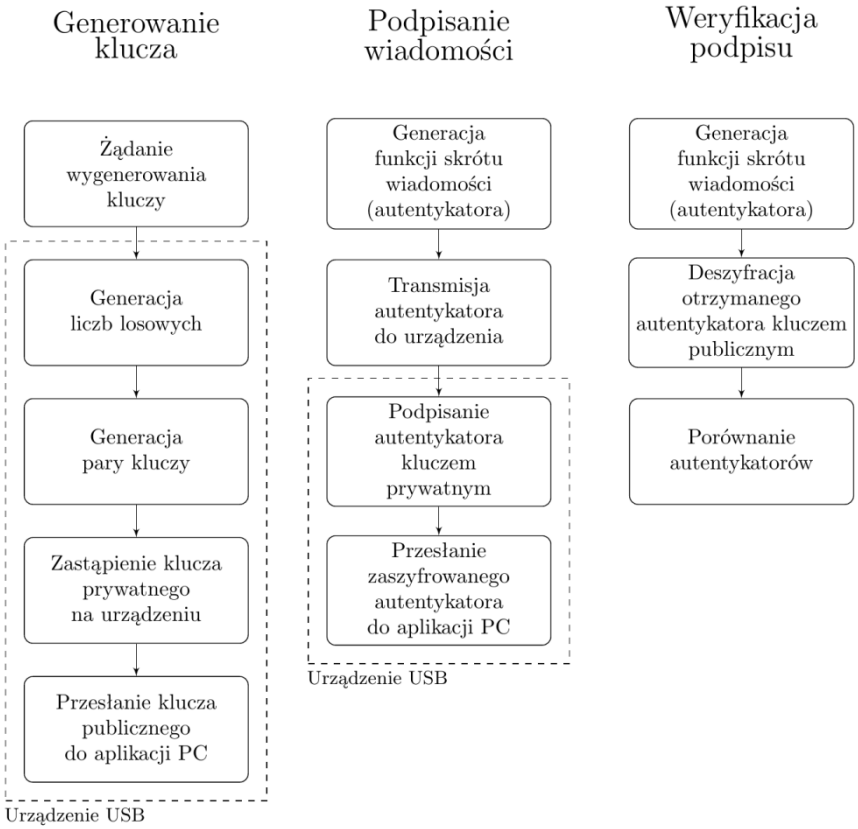
Ze względu na sprzętowy charakter projektu konieczne było napisanie dwóch programów: programu przeznaczonego dla urządzenia USB oraz programu dla komputera osobistego. Aplikacja PC umożliwia użytkownikowi przesłanie do urządzenia USB żądania wygenerowania nowej pary kluczy. Po wykonaniu polecenia, urządzenie w odpowiedzi wysyła wygenerowany klucz publiczny, który użytkownik powinien zachować i przekazać odbiorcom swoich komunikatów. W przypadku podpisu cyfrowego, aplikacji należy dostarczyć plik z wiadomością. W celu uzyskania skrótu wiadomości, na podstawie którego będzie generowany podpis, aplikacja haszuje zawartość pliku funkcją mieszającą SHA-256. Uzyskany ciąg jest przesyłany do urządzenia, które dokonuje operacji podpisu na funkcji skrótu wiadomości, a następnie odsyła do komputera PC. Tak przygotowana wiadomość może zostać udostępniona odbiorcy, który sprawdza jej autorstwo i integralność przy wykorzystaniu klucza publicznego nadawcy oraz funkcji haszującej.

Ponieważ klucz prywatny wygenerowany na mikrokontrolerze powinien być na nim przechowywany aż do czasu wygenerowania nowego klucza, należało umożliwić jego zapamiętanie po pozbawieniu urządzenia zasilania. Wykorzystany mikrokontroler nie posiada wewnętrznej pamięci EEPROM, dlatego konieczna była programowa emulacja.

Aplikacja PC może być wykorzystywana zarówno przez nadawcę, jak również odbiorcę wiadomości, ponieważ umożliwia weryfikację komunikatu.

Poprawność generowania kluczy i operacji z ich wykorzystaniem na mikrokontrolerze kontrolowano podczas pracy nad projektem za pomocą napisanego pomocniczego programu przeznaczonego dla komputera.

Algorytm realizacji poszczególnych funkcji przez programy prezentuje Rysunek 2.



Rysunek 2. Algorytmy realizacji poszczególnych funkcji przez aplikację PC i urządzenie USB.

4.2. Program dla urządzenia USB

Sterowanie urządzeniem odbywa się za pomocą mikrokontrolera. Program, który realizuje mikrokontroler został napisany w języku C. Jest to język na tyle niskopoziomowy, że daje możliwość precyzyjnego sterowanie układem, nie wymagając jednocześnie od programisty dużych nakładów czasowych w celu stworzenia kodu umożliwiającego tę obsługę. Poza tym, większość dostępnych bibliotek (również te udostępniane przez producenta mikrokontrolera) została napisana właśnie w tym języku. Wykorzystano środowisko programistyczne Eclipse z nakładką System Workbench for STM32. Do obsługi mikrokontrolera użyto biblioteki udostępnianej przez producenta: Hardware Abstraction Layer (HAL). Jest stosunkowo dobrze udokumentowana, a ponadto dość popularna wśród programistów mikrokontrolerów STM, co ułatwia rozwiązywanie ewentualnych problemów.

Do konfiguracji mikrokontrolera wykorzystano narzędzie CubeMX udostępniane przez STMicroelectronics. Za jego pomocą dokonano podstawowych inicjalizacji dla peryferiów i zegara systemowego oraz wygenerowano szkielet programu.

Do przesyłu danych wykorzystano masowy tryb transferu (ang. bulk transfer). Charakteryzuje się on kontrolą poprawności przekazywanych danych – błędy transmisji są wykrywane i korygowane poprzez ponowne przesłanie bloku błędnych danych, co jest kluczowe podczas przesyłu danych kryptograficznych. Umożliwia on wykorzystanie pełnej (ang. full speed) lub wysokiej (ang. high speed) szybkości transmisji [6]. Transfer masowy nie ma zagwarantowanego pasma, dlatego nie wykorzystuje się go w systemach, w których regularność dostarczania danych i czas ich przekazywania są krytyczne (np. klawiatury, myszki, sprzęt audio i wideo). W przypadku przedstawianej pracy nie ma takich wymagań; ponadto pozostałe tryby nie spełniłyby innych, ważniejszych wymogów stawianych urządzeniu.

Klasą układu jest Communication Device Class (CDC). Urządzenie tej klasy jest widoczne w systemie operacyjnym jako wirtualny port szeregowy. Emulację interfejsu po stronie mikrokontrolera zapewnia biblioteka HAL.

W przypadku systemu operacyjnego Linux, urządzenia USB CDC są obsługiwane bez wykorzystania dodatkowych sterowników. Dla systemów Windows producent mikrokontrolera udostępnia sterownik dla urządzeń z oprogramowaniem stworzonym przy wykorzystaniu CubeMX.

Wykonywanie algorytmu RSA związane jest z działaniami na liczbach znacznie większych niż rozmiar słowa maszynowego mikrokontrolera. Do tych celów wykorzystano bibliotekę BigDigits, która realizuje wszystkie operacje arytmetyczne potrzebne do implementacji algorytmu. Dodatkowo twórca udostępnia wiele przykładowych programów, które prezentują możliwości biblioteki oraz ułatwiają zapoznanie się z nią. Poszczególne funkcje są dobrze udokumentowane, dzięki czemu ich zrozumienie oraz ewentualne modyfikacje nie sprawiają większych trudności.

W pracy wykorzystano również niewielką bibliotekę EEPROM napisaną przez Nima Askari, która ułatwia emulację pamięci EEPROM na pamięci Flash mikrokontrolerów STM32F103. Dla celów pracy biblioteka została rozbudowana o funkcje, które umożliwiają efektywne wykorzystanie pamięci podczas zapisu i odczytu danych o typach zdefiniowanych przez bibliotekę BigDigits.

4.3. Aplikacja dla komputera osobistego

Program został napisany w językach C/C++, ponieważ biblioteki napisane w tym języku oferują największą elastyczność w obsłudze interfejsu USB od strony hosta (komputera osobistego). Aplikacja posiada interfejs tekstowy. Dzięki temu możliwe jest wygodne dalsze wykorzystanie programu, np. jako element skryptu, który w regularnych odstępach czasu podpisuje, a następnie wysyła komunikaty do odbiorcy.

Wykorzystywana przez program biblioteka Libusb umożliwia komunikację poprzez interfejs USB i obsługuje wiele platform (m.in. Linux, OS X, Windows, Android). Pozwala również na komunikację z poziomu zwykłego użytkownika, bez potrzeby

uruchamiania programu z poziomu użytkownika szczególnie uprzywilejowanego lub administratora.

By zapewnić możliwość weryfikacji wiadomości, konieczne było wykorzystanie biblioteki realizującej działania arytmetyczne na dużych liczbach. Podobnie jak w przypadku programu dla mikrokontrolera, wybrano bibliotekę BigDigits, którą scharakteryzowano w punkcie 4.2.

Do generacji autentykatora dla podpisywanej wiadomości użyto biblioteki Crypto++. Umożliwia ona generowanie kryptograficznej funkcji skrótu SHA-256, wykorzystywanej przez program.

5. Badania czasu wykonywania się algorytmu

Urządzenie zostało poddane testom mającym na celu określenie czasu wykonywania się algorytmu RSA dla operacji generowania nowej pary kluczy oraz operacji podpisu cyfrowego. Pomiar rozpoczyna się bezpośrednio po wywołaniu funkcji realizującej badaną operację i kończy bezpośrednio przed powrotem do programu głównego. Pomiary czasu zostały wykonane dla różnej liczby testów Millera-Rabina. Testy te zostały wykorzystane w algorytmie generowania liczb pierwszych na potrzeby algorytmu RSA w celu wykluczania liczb złożonych. Test nie daje pewności, że liczba jest pierwsza – jeśli jednak powtórzy się go wielokrotnie, prawdopodobieństwo jej pierwszości wzrasta. Jeśli *t*-krotne wykonanie testu z losowymi wartościami *a* nie ujawni, że liczba jest złożona, to statystycznie można ją uważać za liczbę pierwszą z prawdopodobieństwem [8]:

$$p = 1 - \left(\frac{1}{4}\right)^t \tag{12}$$

Tabela 1 przedstawia zmierzone czasy oraz średnie dla 10 prób kolejno przy 25, 50 i 100 testach Millera-Rabina.

Tabela 1. Zmierzone czasy w milisekundach oraz ich średnie dla 10 prób dla różnej liczby testów Millera-Rabina.

	Liczba testów Millera-Rabina		
	25	**50**	**100**
Uzyskane czasy poszczególnych prób [ms]	33968	89435	112134
	25875	53259	111510
	41577	44241	102623
	55056	44465	113140
	72276	62620	101937
	67829	60874	108423
	29112	83383	117096
	49408	71537	106283
	46738	70174	91151
	49980	43095	125966
Średnia	**47181.9**	**62308.3**	**109026.3**

Na podstawie wyników pomiarów zadecydowano, że program będzie wykonywał 50 testów Millera-Rabina. Wybrana wartość daje bardzo dużą pewność pierwszości liczby (szansa na liczbę złożoną rzędu $8x10^{-31}$), a jednocześnie średni czas generowania kluczy wynosi nieznacznie powyżej minuty.

Tabela 2 przedstawia czasy 10 prób generowania podpisu cyfrowego dla 256-bitowego skrótu wiadomości.

Tabela 2. Czas generowania podpisu cyfrowego dla 256-bitowej funkcji skrótu.

lp.	Czas [ms]
1	833
2	832
3	832
4	833
5	833
6	832
7	832
8	832
9	833
10	832
Średnia	**832.4**

6. Podsumowanie

Zaprojektowano i zbudowano urządzenie realizujące operacje generowania 1024-bitowych kluczy oraz podpisywania komunikatu. Urządzenie wykorzystuje do tych celów algorytm RSA i komunikuje się z komputerem osobistym za pomocą interfejsu USB. Napisano również program dla komputera PC, który umożliwia komunikację ze strony hosta. Opisany system zapewnia bezpieczne przechowywanie klucza prywatnego – jest on tworzony, a następnie zapisany i wykorzystywany wyłącznie na zewnętrznym urządzeniu USB.

Otrzymane czasy wskazują na generowanie 1024-bitowych kluczy w ciągu około minuty, natomiast podpisanie wiadomości trwa niecałą sekundę. Są to czasy umożliwiające wygodne korzystanie z zaprezentowanego rozwiązania. Czas generowania klucza jest zauważalny dla użytkownika, jednak operacja generowania nowej pary kluczy jest zazwyczaj wykonywana znacznie rzadziej niż operacja podpisu cyfrowego.

Należy zaznaczyć, że przedstawiona implementacja obejmuje wyłącznie podpisywanie i weryfikację komunikatu. Dalsze prace mogłyby obejmować realizację algorytmu RSA również dla potrzeb szyfrowania oraz połączonych operacji szyfrowania i podpisu.

LITERATURA

1. STALLINGS W.: AutoLISP czyli programowanie AutoCADa. Helion, Gliwice 1995.
2. QIAO, G. LAM, K.: RSA Signature Algorithm for Microcontroller Implementation. Smart Card Research and Applications: Third International Conference, CARDIS'98, Louvain-la-Neuve, Belgium, September 14-16, 1998. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, 353–356.
3. DIFFIE, W.: The first ten years of public-key cryptography. Proceedings of the IEEE 76.5, (1988), 560-577.
4. DIFFIE, W. HELLMAN, M. E.: Multiuser Cryptographic Techniques. Proceedings of the June 7-10, 1976, National Computer Conference and Exposition. New York (1976), 109-122.
5. RIVEST, R. L. SHAMIR, A. Adleman, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Commun. ACM 21.2(1978), 120-126.
6. MIELCZAREK, W.: USB: uniwersalny interfejs szeregowy : kompletny opis architektury systemu komputerowego opartego na złączu Universal Serial Bus. Helion, Gliwice 2005.
7. USB 2.0 Specification. 2000.
8. RABIN M. O.: Probabilistic algorithm for testing primality. Journal of number theory, 12(1), (1980), 128-138.