Prototyp bezpiecznego mikroserwera przemysłowego dla rozproszonego systemu kontrolno-pomiarowego

Dariusz Rzońca, Andrzej Stec

Katedra Informatyki i Automatyki, Politechnika Rzeszowska

Streszczenie: Problem zabezpieczenia transmisji w rozproszonych systemach kontrolno-pomiarowych jest bardzo istotny. Ograniczone zasoby sprzętowe często wymuszają opracowanie dedykowanych rozwiązań. W artykule przedstawiono propozycję prototypowego mikroserwera, zapewniającego bezpieczną komunikację ze zdalną stacją operatorską.

Słowa kluczowe: bezpieczny serwer, bezpieczny transfer danych, system kontrolno-pomiarowy

W ramach prac prowadzonych w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej opracowany został prototyp mikroserwera, którego zadaniem jest zapewnienie bezpiecznej transmisji danych w sieci internet pomiędzy zdalnym użytkownikiem i niewielkim przemysłowym systemem kontrolno-pomiarowym.

1. Wprowadzenie

Nowoczesne systemy kontrolno-pomiarowe coraz powszechniej korzystają z infrastruktury sieci komputerowych [1]. Nie są to już jedynie moduły I/O podłączone do układu mikroprocesorowego, ale zespoły inteligentnych czujników lub sterowników realizujących zadane funkcjonalności, z możliwością wymiany danych za pośrednictwem sieci. Elementy składowe takich systemów mogą być rozproszone na dużym obszarze, z dala od jednostki centralnej i mogą realizować różne zadania w zależności od bieżącej potrzeby. Coraz częściej nieodzownym elementem takiego systemu jest również serwer WWW gromadzący dane z systemu i udostępniający je przez Internet. Urządzenia te często nie mają dużej mocy obliczeniowej z uwagi na koszt oraz jak najmniejsze zużycie energii. Niemniej jednak kwestie bezpieczeństwa przesyłanych danych nie mogą być pomijane, zwłaszcza wtedy, gdy operator zdalnie musi wydawać polecenia wpływające na działanie systemu.

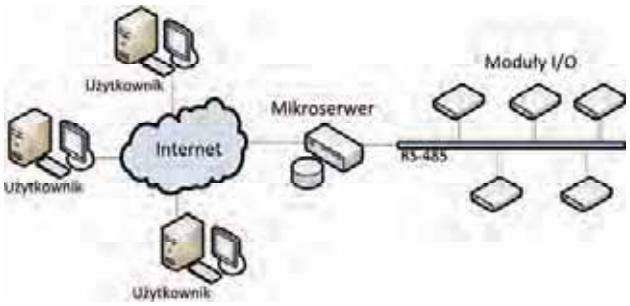
Przy korzystaniu z infrastruktury publicznej lub połączeń bezprzewodowych, szczególnie w miejscach, do których mają dostęp osoby postronne, konieczne staje się korzystanie z połączeń szyfrowanych. Niestety, bardzo często sterowniki w niewielkich systemach kontrolno-pomiarowych mają silnie ograniczone zasoby, a ich wydajność obliczeniowa nie pozwala na stosowanie algorytmów wykorzystywanych w rozwiązaniach znanych z komputerów PC. Ponadto, różnego rodzaju nawet niewielkie zakłócenia w funkcjonowaniu sieci czy choćby znaczne opóźnienia przy przesyłaniu danych mogą być bardzo szkodliwe dla działania takich systemów. Konieczne staje się więc dokładne analizowanie mechanizmów wymiany danych, tak aby projektowany system spełniał przyjęte wymagania czasowe [2]. Rozważając problematykę zabezpieczenia transmisji w rozproszonych systemach kontrolno-pomiarowych, bądź rozproszonych systemach sterowania DCS (D*istributed Control System*) [3], można wyodrębnić komunikację poziomą oraz pionową [4], a następnie osobno rozpatrywać mechanizmy bezpieczeństwa możliwe do zastosowania na poszczególnych poziomach. Przykładowo w [5] zaproponowano metodę rozwiązania problemu braku uwierzytelniania w protokole Modbus, często stosowanym w sieciach polowych (*fieldbus*). Niniejsza praca poświęcona jest głównie zagadnieniu zabezpieczenia transmisji pomiędzy centralnym mikroserwerem WWW a przeglądarką internetową na stacji operatorskiej. Aspekt zabezpieczenia transmisji podczas akwizycji danych z modułów obiektowych wewnątrz systemu kontrolno-pomiarowego nie jest tu rozważany.

2. Architektura systemu

W proponowanym rozwiązaniu zakłada się, że system kontrolno-pomiarowy składa się z jednostki centralnej odpowiedzialnej za koordynację jego funkcjonowania oraz wielu modułów obiektowych (I/O) realizujących odczyt danych z czujników i sterowanie elementami wykonawczymi. Komunikacja odbywa się za pośrednictwem magistrali RS-485, przy wykorzystaniu jednego z popularnych protokołów *master-slave*, np. Modbus RTU. System taki uzupełniany jest o niewielki serwer gromadzący dane otrzymane z jednostki centralnej i prezentujący je na stronie internetowej. Alternatywnym, lepszym rozwiązaniem jest jednak wbudowanie funkcjonalności serwera w jednostkę centralną lub przejęcie funkcji jednostki centralnej przez serwer.

W rozwiązaniu zaprezentowanym na rys. 1 każdy z modułów I/O może być samodzielnym sterownikiem wykonującym własny program. Mikroserwer, działający pod kontrolą systemu operacyjnego czasu rzeczywistego pełni zarówno funkcję jednostki centralnej odpytującej urządzenia podrzędne (moduły) jak i funkcję serwera danych podłączonego do sieci Internet. W ten sposób staje się on jednocześnie koncentratorem danych pomiarowych, źródłem danych procesowych do realizacji sterowania

przez moduły obiektowe, a także bazą danych krótkookresowych dla zewnętrznych systemów.



Rys. 1. Architektura systemu
Fig. 1. System architecture

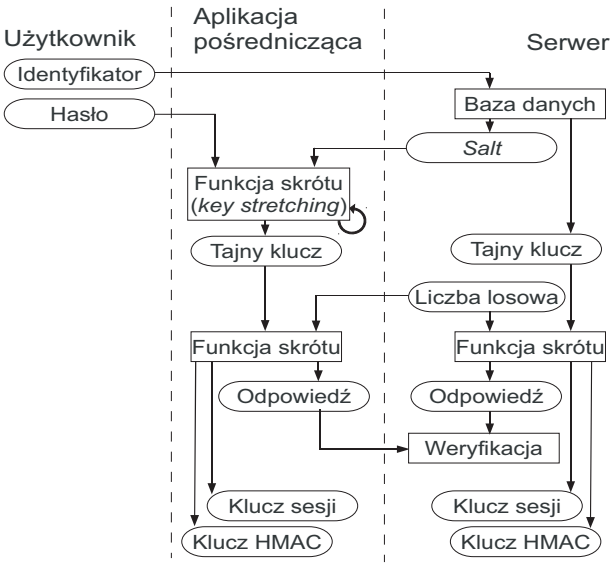
Mikroserwer bazuje na rozwiązaniu sprzętowym wykorzystującym mikrokontroler ARM7 (LPC2468) i systemie operacyjnym FreeRTOS [6, 7] z oprogramowaniem uIP do obsługi stosu protokołów TCP/IP. Opisywany system stanowi rozwinięcie zaproponowanego w [8], gdzie moduły bazowały na płytach ewaluacyjnych AT91SAM7S-EK. Przyjęto, że jednocześnie z zasobów mikroserwera nie będzie korzystało więcej niż kilku klientów, a obciążenie wynikające z ilości danych pobieranych jednorazowo nie będzie duże (do kilkuset KB). Od momentu nawiązania bezpiecznego połączenia cała transmisja jest szyfrowana, a próby ingerencji w przesyłane dane powodują automatyczne zerwanie połączenia i konieczność ponownego uwierzytelnienia użytkownika. Zdalny dostęp do mikroserwera mają jedynie użytkownicy uprzednio zarejestrowani przez administratora, których login i zakodowane hasło zapisane są w wewnętrznej bazie. Postać przechowywanego hasła nie pozwala na jego użycie nawet w przypadku jego przechwycenia. Pierwsze uruchomienie mikroserwera wymaga zmiany domyślnego hasła administratora, podobnie jak w typowym sprzęcie sieciowym. Komunikacja z mikroserwerem odbywa się za pośrednictwem dowolnej przeglądarki internetowej lub innego programu obsługującego protokół http. Z powodu ograniczonych zasobów sprzętowo-programowych mikroserwera zastosowane algorytmy szyfrujące ograniczają się do kryptografii symetrycznej, co opisano w rozdziale 3. Wymaga to dodatkowo uruchomienia na komputerze klienta specjalizowanej aplikacji pośredniczącej zajmującej się szyfrowaniem i deszyfrowaniem komunikatów przesyłanych z i do przeglądarki internetowej, szerzej przedstawionej w rozdziale 4.

3. Mechanizmy bezpieczeństwa

Problematyka zabezpieczenia dostępu do rozproszonych systemów kontrolno-pomiarowych jest istotna i dość złożona [9–11]. Należy podkreślić, że niekiedy specyfika zastosowanych urządzeń wymaga innych rozwiązań niż typowo stosowane w dużych systemach [12], głównie ze względu na niewystarczające zasoby sprzętowe. Implikuje to potrzebę opracowywania rozwiązań dedykowanych, jak np. zaproponowano w [13]. Podobnie w opisywanym mikroserwerze niewielka moc obliczeniowa mikroprocesora wymusiła rezygnację z kryptografii asymetrycznej i konieczność oparcia zabezpieczeń wyłącznie na algorytmach symetrycznych. Decyzja taka pociągnęła za sobą konieczność przygotowania wspomnianej specjalnej aplikacji pośredniczącej realizującej zabezpieczenie transmisji, szerzej opisanej w rozdziale 4. Prezentowane rozwiązanie jest rozwinięciem zaprezentowanego w [14].

Mechanizmy zabezpieczania transmisji powinny zapewniać uwierzytelnianie użytkownika (tj. gwarantować, że użytkownik jest tym, za kogo się podaje), poufność (ochronę przed odczytem informacji przez osoby niepowołane) i integralność (pewność, że wiadomość dotarła od nadawcy do odbiorcy w postaci niezmienionej) transmisji. W opisywanym rozwiązaniu uwierzytelnianie oparte jest na identyfikatorze i haśle. Po stronie serwera zapisywany jest jedynie losowy *salt* i tajny klucz będący *hashem* hasła (wynik działania na hasło i *salt* pewną jednokierunkową funkcją skrótu), zgodnie z PBKDF2 [15], nie zaś hasło w postaci jawnej. Dzięki temu nie ma możliwości odczytania hasła, można jedynie sprawdzić czy dane hasło zgadza się z zapisanym. Proces uwierzytelnienia oparty jest na protokole typu wyzwanie-odpowiedź (*challenge-response*) opisanym w [16]. Zaproponowana implementacja przedstawiona jest na rys. 2.

Użytkownik wprowadza w aplikacji pośredniczącej swój identyfikator i hasło. Identyfikator w postaci jawnej przesyłany jest na serwer. Po stronie serwera z bazy danych odczytywany jest tajny klucz i powiązany z nim *salt* dla danego identyfikatora. Losowana jest także liczba stanowiąca wyzwanie (*challenge*). Liczba ta wraz z *saltem* odsyłana jest do aplikacji pośredniczącej. W aplikacji na

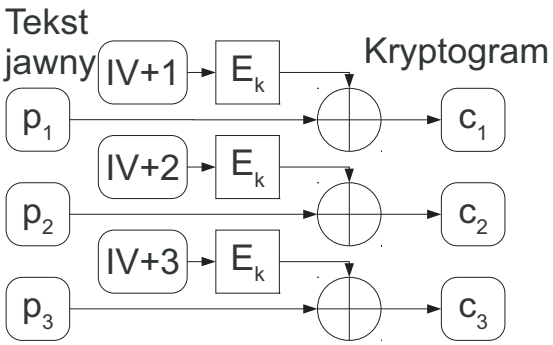


Rys. 2. Implementacja protokołu *challenge-response*
Fig. 2. Implementation of challenge-response protocol

podstawie *saltu* i wprowadzonego przez użytkownika hasła poprzez wielokrotne użycie funkcji skrótu (*key stretching*) generowany jest tajny klucz, identyczny z zapisanym w bazie na serwerze. Klucz ten wraz z liczbą losową po

przejściu przez funkcję skrótu tworzy odpowiedź, jak również jednorazowe klucze sesji i HMAC. Odpowiedź przesyłana jest na serwer, gdzie następuje jej weryfikacja. W ten sposób niezależnie po obu stronach, tj. w aplikacji i na serwerze, wygenerowane zostają klucze służące zabezpieczaniu dalszej transmisji.

Jako algorytm szyfrujący wykorzystano AES-128 [17], natomiast jako funkcję skrótu SHA-256 [18]. Blokowe algorytmy szyfrujące (jak AES) wykorzystywane są w pewnych trybach, definiujących sposób otrzymywania kolejnych bloków kryptogramu w zależności od tekstu jawnego, hasła, ale także np. numeru bloku bądź zawartości bloku poprzedniego. Trywialny tryb elektronicznej książki kodowej ECB (*Electronic CodeBook*), w którym kryptogram zależy jedynie od tekstu jawnego i hasła nie zapewnia elementarnego poziomu bezpieczeństwa, gdyż jednakowym wiadomościom odpowiadają jednakowe kryptogramy. Gdy identyczne wiadomości często się powtarzają (co jest typowe w systemach sterowania) intruz mógłby w łatwy sposób odgadnąć znaczenie poszczególnych kryptogramów, bez znajomości hasła. W opisywanym rozwiązaniu wykorzystano tryb licznikowy CTR (*Counter*) szyfru, pokazany na rysunku 3. Przyjęto tu oznaczenia: c_i – i-ty kryptogram, p_i – i-ty tekst jawny, IV – wektor inicjalizacyjny, E_k – funkcja szyfrująca przy kluczu k.

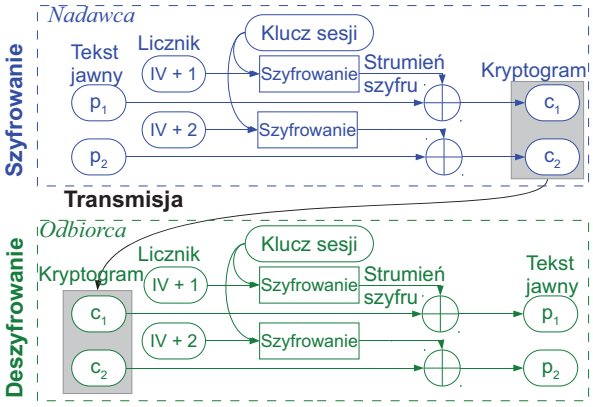


Rys. 3. Tryb licznikowy szyfru blokowego
Fig. 3. Counter mode of block cipher

Tryb licznikowy szyfru działa wg wzorów:

$$c_i = E_k(IV+i)\,\mathrm{XOR}\; p_i$$
$$p_i = E_k(IV+i)\,\mathrm{XOR}\; c_i$$

Tryb ten *de facto* zmienia szyfr blokowy w szyfr strumieniowy. Kolejne wartości licznika wraz z wektorem inicjalizacyjnym są szyfrowane algorytmem AES przy użyciu klucza sesji. Wynik szyfru stanowi strumień szyfrujący, przy pomocy którego funkcją XOR zmieniany jest tekst jawny w wiadomość zaszyfrowaną. Należy zauważyć, że proces dekryptażu przebiega identycznie, taki sam strumień szyfrujący wygenerowany u odbiorcy (również przy użyciu funkcji szyfrującej, a nie deszyfrującej z AES) poprzez funkcję XOR deszyfruje kryptogram. Proces ten przedstawiono na rys. 4.



Rys. 4. Proces szyfrowania i deszyfrowania
Fig. 4. Encrypting and decrypting process

Wykorzystanie algorytmu AES w trybie CTR gwarantuje poufność, ale nie zapewnia integralności transmisji. Intruz podczas ataku *man-in-the-middle* nie byłby w stanie odczytać treści zaszyfrowanej wiadomości, ale mógłby ślepo modyfikować kryptogram zmieniając znaczenie przekazu, a odbiorca nie byłby w stanie odróżnić wiadomości prawidłowej od spreparowanej. Co więcej, gdyby intruz był w stanie odgadnąć prawdziwą treść wiadomości, to mógłby na tej podstawie uzyskać strumień szyfrujący i podsunąć odbiorcy dowolną swoją wiadomość. Warto zauważyć, że w rozproszonych systemach automatyki często wiadomości wymieniane są w bardzo przewidywalny sposób (np. wysłanie polecenia odczytu konfiguracji bezpośrednio po nawiązaniu połączenia), co znacznie zwiększałoby prawdopodobieństwo wspomnianego ataku.

Aby zapewnić kontrolę integralności transmisji w przypadku, gdy nie gwarantuje tego zastosowany tryb szyfru blokowego, konieczne jest wykorzystanie osobnego mechanizmu. Oczywiście nie chodzi tu o proste sumy kontrolne, które pozwalają na wykrycie przypadkowej zmiany spowodowanej przekłamaniami podczas transmisji, ale o kod, którego wartości dla spreparowanej wiadomości intruz nie będzie w stanie obliczyć nie znając klucza. W opisywanym rozwiązaniu wykorzystano kod HMAC (*Hash-based Message Authentication Code*) [19].

Zaproponowane rozwiązanie przeznaczone jest dla niewielkich rozproszonych systemów automatyki, szczególnie istotne więc jest rozpatrywanie jego parametrów czasowych. Opisywany mikroserwer typowo stanowi centralny węzeł takiego systemu, zajmując się akwizycją danych z podłączonych modułów lub też sterowaniem nadrzędnym. Można przykładowo rozważać przypadek, gdy poszczególne moduły podrzędne odpytywane są cyklicznie w pewnym protokole *master-slave* (transmisja inicjowana jest wyłącznie przez mikroserwer). System taki rozpatrywany był w [20]. Jednym z rozważanych parametrów było średnie opóźnienie wartości zmiennych przesyłanych klientowi przez serwer t_{client_lat}. Dane, cyklicznie odczytywane przez serwer przechowywane są w lokalnej pamięci pośredniej serwera (*cache*). Klient sieciowy odczytując zmienne z serwera otrzymuje wartości przechowywane w *cache*, wraz z ich stemplem czasowym. Opóźnienie zależy od ilości podłączonych modułów n, czasu potrzebnego na

wykonanie programu sterowania w każdym z modułów t_{prog} i opóźnień sieciowych. Czas t_{prog} jest krótszy niż założony czas cyklu t_{cycle} i może zmieniać się w kolejnych cyklach. W celu ustalenia jak t_{client_lat} zależy od t_{prog}, tj. od złożoności programu sterowania, zaproponowano model oparty na hierarchicznej czasowej kolorowanej sieci Petriego (HTCPN – *Hierarchical Timed Coloured Petri Nets*) i przeprowadzono jego badania symulacyjne. Wyprowadzono także wzór na t_{client_lat} w opisywanym systemie. Przyjmując oznaczenia t_{bus} – czas transmisji ramki podczas odpytywania modułów, t_{msg} – czas przetwarzania wiadomości przez moduł, t_{encr} – czas szyfrowania, t_{PC_net} – czas transmisji do przeglądarki otrzymano:

$$t_{client\_lat} = \frac{nt_{prog}^2}{4t_{cycle}} + (n+1)t_{bus} + \frac{n}{2}t_{msg} + t_{encr} + t_{PC\_net}$$

Innym celowym do rozważenia zagadnieniem związanym z parametrami czasowymi systemu jest wpływ utraty niektórych pakietów podczas transmisji na czas nawiązania połączenia w protokole *challenge-response*. Analizę taką przeprowadzono w [21]. Utrata pakietu wymusza ponowną próbę nawiązania połączenia. Zakładając stałe prawdopodobieństwo utraty pojedynczego pakietu P_u proces nawiązywania połączenia możemy traktować jako ciąg niezależnych prób Bernoulliego, wówczas zmienna losowa podająca w której z prób udało się nawiązać połączenie ma rozkład geometryczny. Przyjmując oznaczenia t_{tout} – maksymalny czas oczekiwania na odpowiedź, t_{net} – czas transmisji pojedynczego pakietu, wyprowadzono tam następujący wzór na średni czas nawiązania połączenia t_{conn}.

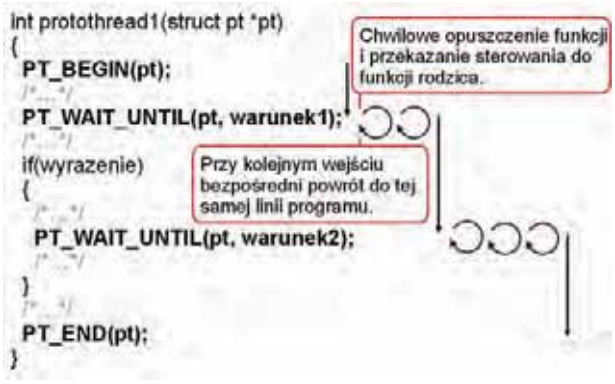
$$t_{conn} = \frac{1-(1-P_u)^3}{(1-P_u)^3}t_{tout} + 3t_{net}$$

Podaną zależność teoretyczną porównano z wynikami badań eksperymentalnych. Wraz ze wzrostem prawdopodobieństwa utraty pojedynczego pakietu gwałtownie rośnie średni czas potrzebny na nawiązanie połączenia, co w pewnym stopniu może ograniczać obszar stosowalności proponowanego rozwiązania. Należy jednak zauważyć, że w typowych sieciach komunikacyjnych spełnienie przedstawionych wymagań nie stwarza problemów.

4. Implementacja

Oprogramowanie prezentowanego mikroserwera bazuje na systemie operacyjnym czasu rzeczywistego FreeRTOS [6, 7], przystosowanym do wymogów sprzętowych zastosowanego mikrokontrolera, uzupełnionym o stos uIP [22] do obsługi komunikacji ethernetowej. Stos uIP jest stosunkowo prostym stosem TCP/IP, powszechnie stosowanym w systemach wbudowanych wykorzystujących mikrokontrolery o małej mocy obliczeniowej i o małym zapotrzebowaniu dotyczącym pamięci RAM. Zawiera jednak w sobie wszystkie podstawowe funkcjonalności niezbędne do obsługi komunikacji sieciowej. Obsługuje m.in. protokoły ARP, IP, ICMP oraz TCP. Dodatkowym wsparciem są przykładowe aplikacje, np. prosty serwer WWW, udostępniane wraz ze stosem. Choć uIP sam w sobie nie ma wbudowanych żadnych mechanizmów wspomagających szyfrowanie, to jednak z uwagi na udostępnienie pełnego kodu źródłowego na liberalnej licencji BSD-style (*open source*) pozwala na uzupełnienie stosu o niezbędne funkcje.

Stos uIP został opracowany z wykorzystaniem dość specyficznego mechanizmu zwanego protothreadami [23]. Można go uznać za pewnego rodzaju styl programowania przypominający programowanie oparte na wątkach, charakteryzujący się linowo-strukturalną formą, mający zastosowanie przy implementacji prostych automatów stanów opartych na zdarzeniach. Sam w sobie jest jedynie sprytnym wykorzystaniem makrodefinicji i instrukcji switch języka C, sprawiając, że zwykły program zachowuje się tak, jakby działał pod kontrolą systemu operacyjnego wywołującego cyklicznie kolejne wątki w oczekiwaniu na zakończenie się bieżącej operacji (rys. 5).



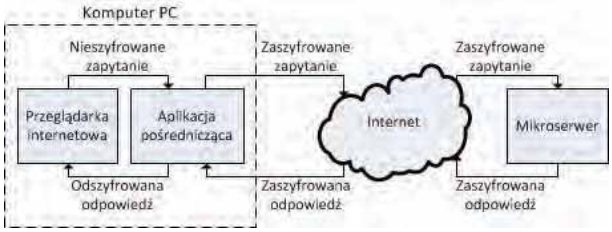
Rys. 5. Właściwości protothreadów
Fig. 5. Features of protothreads

Korzystanie z mechanizmu protothreadów, zwłaszcza na początku, może nie być sprawą prostą. Wymaga bowiem od autora pełnej kontroli nad swoim programem, zwłaszcza różnego rodzaju znaczników ustawianych w funkcjach wewnętrznych. Gdy obsługiwanych jest równocześnie kilka operacji wymagających wzajemnej synchronizacji bardzo łatwo jest coś przeoczyć.

Jednym z elementów opracowanego rozwiązania jest napisana przez autorów artykułu biblioteka funkcji kryptograficznych. Bazując na ogólnodostępnych, elementarnych funkcjach do szyfrowania AES-128, obliczania funkcji skrótu SHA-256 i kodu HMAC przygotowana została biblioteka pozwalająca na obsługę bezpiecznego połączenia w sposób opisany w rozdziale 3. Z uwagi na specyfikę przesyłanych komunikatów niezbędnym było także uzupełnienie stosu uIP o fragmenty kodu dotyczące obsługi bufora nadsyłanych danych. Największe zmiany dotyczyły jednak przykładowego serwera WWW, który należało uzupełnić o możliwość nawiązywania połączenia w protokole *challenge-response* oraz gruntownie zmodyfikować kod do obsługi protokołu http.

Ponieważ opracowane rozwiązanie wykorzystuje mechanizm szyfrowania, który nie jest standardowo wbudowany w przeglądarki internetowe, dlatego wymiana informacji pomiędzy komputerem klienta a mikroserwerem wymaga uprzedniego uruchomienia specjalnej aplikacji pośredniczącej. Jej zadaniem jest nawiązanie bezpiecznego połączenia, a potem szyfrowanie i deszyfrowanie przesyła-

nych komunikatów (rys. 6). Aplikacja pośrednicząca nie musi być instalowana na komputerze, ale w przypadku stosowania zapory wymaga akceptacji użytkownika do korzystania z połączenia sieciowego.



Rys. 6. Przepływ danych pomiędzy klientem i serwerem
Fig. 6. Client-server data flow

Aplikacja pośrednicząca jest dostępna dla przeglądarki pod adresem localhosta (127.0.0.1), na porcie wybranym przez użytkownika (np. 32000). W celu połączenia się z nią, po uprzednim uruchomieniu aplikacji pośredniczącej, w pasku adresu url przeglądarki internetowej należy wpisać http://localhost:32000. W odpowiedzi, w oknie powinna pojawić się strona powitalna prosząca o podanie adresu IP mikroserwera, a także loginu i hasła użytkownika. Przesłanie danych do aplikacji pośredniczącej rozpoczyna proces nawiązywania bezpiecznego połączenia z mikroserwerem i uwierzytelniania użytkownika. Raz nawiązane połączenie jest aktywne do momentu wylogowania się użytkownika, upłynięcia czasu związanego z brakiem aktywności lub w przypadku wystąpienia błędu w transmisji danych. Aplikacja pośrednicząca nie interpretuje przesyłanych komunikatów, ale jeżeli otrzymany szyfrogram jest niepoprawny, wtedy następuje automatyczne zerwanie połączenia, a użytkownik musi być ponowne uwierzytelniony. Próby logowania się do mikroserwera bez korzystania z aplikacji pośredniczącej, w zależności od ustawień konfiguracyjnych, mogą być ignorowane lub skutkować odesłaniem strony internetowej informującej o konieczności uruchomienia aplikacji. W celu uniknięcia ryzyka związanego możliwością podszywania się pod mikroserwer aplikacja pośrednicząca powinna być pobierana jedynie z zaufanego serwera lub innego bezpiecznego nośnika. Do obowiązku użytkownika należy także zadbanie, aby na komputerze klienta nie było uruchomionego oprogramowania umożliwiającego przechwytywanie nieszyfrowanych danych pomiędzy przeglądarką internetową a aplikacją pośredniczącą. Szczególnie dotyczy to procesu nawiązywania połączenia, w trakcie którego przesyłane są login i hasło użytkownika. Aby ograniczyć ryzyko związane z możliwością złamania hasła warto jest stosować ogólne zasady dotyczące jego długości czy używanych znaków alfanumerycznych.

Aplikacja pośrednicząca może być uruchamiana z widocznym (rys. 7) lub ukrytym interfejsem graficznym. W zależności od ustawień konfiguracyjnych (wartość *infoLevel*) ilość wyświetlanych informacji może być mniejsza lub większa. Pojawiające się w oknie komunikaty mogą informować np. jedynie o zalogowaniu lub wylogowaniu się użytkownika, ale także o rozmiarze każdej paczki danych przesłanych pomiędzy przeglądarką, aplikacją pośredniczącą i mikroserwerem.

Rys. 7. Aplikacja pośrednicząca
Fig. 7. Intermediary application

5. Podsumowanie

Małe rozproszone systemy kontrolno-pomiarowe wykorzystujące komunikację sieciową (ethernetową) często wyposażone są w urządzenia z mikrokontrolerami o mocy obliczeniowej niewystarczającej do stosowania bezpiecznych rozwiązań spotykanych w komputerach klasy PC. Jak zostało pokazane wymagany poziom bezpieczeństwa może być jednak osiągnięty przez zastosowanie dedykowanego rozwiązania. Wystarczającym jest korzystanie z algorytmów szyfrowania symetrycznego oraz uzgadnianie klucza sesji za pomocą protokołu *challenge-response*. Aby możliwe było używanie zwykłej przeglądarki internetowej, konieczne jest korzystanie z aplikacji pośredniczącej uruchamianej na komputerze PC klienta. Przeprowadzone symulacje i testy prezentowanego systemu pokazują, że opóźnienia wprowadzane przez algorytm szyfrowania i deszyfrowania są akceptowalne nawet dla mikroserwera z mikrokontrolerem o silnie ograniczonych zasobach sprzętowo-programowych.

W badaniach wykorzystano aparaturę naukowo-badawczą zakupioną w ramach projektu pt. „Budowa, rozbudowa i modernizacja bazy naukowo-badawczej Politechniki Rzeszowskiej" współfinansowanego ze środków Unii Europejskiej w ramach Regionalnego Programu Operacyjnego Województwa Podkarpackiego na lata 2007-2013, Priorytet I. Konkurencyjna i Innowacyjna Gospodarka, Działanie 1.3 Regionalny system innowacji.

Bibliografia

1. Kwiecień R., *Komputerowe systemy automatyki przemysłowej*, Helion, Gliwice 2012.
2. Kwiecień A., *Analiza przepływu informacji w komputerowych sieciach przemysłowych*, „Studia Informatica", Vol. 23, No. 1 (47), 2002.
3. Feng-Li Lian, Moyne J., Tilbury D., *Network Design Consideration for Distributed Control Systems*, „IEEE Transactions on Control Systems Technology" Vol. 10, No. 2, 2002, 297–307.
4. Gaj P., Jasperneite J., Felser M., *Computer Communication within Industrial Distributed Environment* -

a Survey, „IEEE Transactions on Industrial Informatics" (w druku).
5. Liao G.Y., Chen Y.J., Lu W.C, Cheng T.C, *Toward Authenticating the Master in the Modbus Protocol*, „IEEE Transactions on Power Delivery" Vol. 23, No. 4, Oct 2008, 2628–2629.
6. Barry R., *Using the FreeRTOS Real Time Kernel – A Practical Guide*, 2010.
7. [http://www.freertos.org/] – strona internetowa projektu FreeRTOS (listopad 2012).
8. Rzońca D., Stec A., Trybus B., *Data Acquisition Server for Mini Distributed Control System*, [w:] Kwiecien A., Gaj P., and Stera P. (Eds.), „Computer Networks 2011, Communications in Computer and Information Science", Vol. 160, Springer-Verlag Berlin Heidelberg 2011, 398–406.
9. Cheminod M., Pironti A., Sisto R., *Formal Vulnerability Analysis of a Security System for Remote Fieldbus Access*, „IEEE Transactions on Industrial Informatics", Vol. 7, No. 1, Feb 2011, 30–40.
10. Liderman K., Zieliński Z. *Ochrona informacji w połączonych sieciach przemysłowych i teleinformatycznych*, [w:] Huzar Z., Mazur Z. (red.) *Zagadnienia bezpieczeństwa w systemach informacyjnych*, Wydawnictwa Komunikacji i Łączności, Warszawa 2008.
11. Mendyk-Krajewska T., *Podatność na ataki sieci przemysłowych*, [w:] Trybus L., Samolej S. (red.), *Projektowanie, analiza i implementacja systemów czasu rzeczywistego*, Wydawnictwa Komunikacji i Łączności, Warszawa 2011.
12. Valenzano A., Durante L., Cheminod M., *Review of Security Issues in Industrial Networks*, „IEEE Transactions on Industrial Informatics" (w druku).
13. Szymczyk P., Szymczyk M., Szwed P., Rogus G., *System wbudowany z bezpiecznym asynchronicznym interfejsem WWW*, [w:] Trybus L., Samolej S. (red.) *Metody wytwarzania i zastosowania systemów czasu rzeczywistego*, Wydawnictwa Komunikacji i Łączności, Warszawa 2010.
14. Rzońca D., Stec A., *Small Prototype Acquisition System with Secure Remote Data Access*, „Annales UMCS Informatica AI" Vol. XI, No. 3 (2011), 87–100.
15. Turan M.S., Barker E., Burr W., Chen L., *Recommendation for Password-Based Key Derivation Part 1: Storage Applications*. „National Institute of Standards and Technology Special Publications", Dec. 2010, 800–132, [http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf].
16. Menezes A.J., van Oorschot P.C., Vanstone S.A., *Kryptografia stosowana*, Wydawnictwa Naukowo-Techniczne, Warszawa 2005.
17. *Advanced Encryption Standard (AES)*, „Federal Information Processing Standards Publications" 197, National Institute of Standards and Technology, Nov. 2001, [http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf].
18. *Secure Hash Standard (SHS)*, „Federal Information Processing Standards Publications" 180-3, National Institute of Standards and Technology, Oct. 2008. [http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf].
19. *The Keyed-Hash Message Authentication Code (HMAC)*, „Federal Information Processing Standards Publications" 198-1, National Institute of Standards and Technology, Jul. 2008, [http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1-final.pdf].
20. Rzońca D., Stec A., Trybus B., *Secure Web Access to Mini Distributed Control System*, Półrocznik AGH Automatyka/Automatics (w druku).
21. Rząsa W., Rzońca D., Stec A., Trybus B., *Analysis of Challenge-response Authentication in a Networked Control System*, [w:] Kwiecien A., Gaj P., and Stera P. (Eds.), „Computer Networks 2012, Communications in Computer and Information Science" 291, Springer-Verlag Berlin Heidelberg 2012, 271–279.
22. Dunkels A. *Protothreads. Reference manual v.1.4*. 2006. [http://dunkels.com/adam/pt/download.html]
23. Dunkels A., Schmidt O., Voigt T., Ali M., *Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems*, [w:] *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06)*, ACM, New York, 2006.

Prototype of secure Industrial microserver for distributed control and measurement system

Abstract: Security problems in distributed control and measurement systems are very important. Limited hardware resources often necessitate development of dedicated solutions. The paper describes a prototype microserver, providing secure communication with remote control station.

Keywords: secure server, secure data transfer, control and measurement system

dr inż. Dariusz Rzońca

Licencjat matematyki (Uniwersytet Rzeszowski 2002), magister inżynier informatyki (Politechnika Rzeszowska 2004), doktor nauk technicznych w dyscyplinie informatyka, specjalność przemysłowe systemy informatyki (Politechnika Śląska 2012). Asystent w Katedrze Automatyki i Informatyki Politechniki Rzeszowskiej od 2004 roku. Jego zainteresowania naukowe koncentrują się na kolorowanych sieciach Petriego oraz zagadnieniach związanych z komunikacją w systemach automatyki.

e-mail: drzonca@prz-rzeszow.pl

dr inż. Andrzej Stec

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (1993). Stopień doktora nauk technicznych uzyskał w 2004 r. Obecnie adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Zajmuje się zagadnieniami dotyczącymi systemów wbudowanych i rozproszonych systemów sterowania.

e-mail: astec@prz-rzeszow.pl