

Projekt z przedmiotu „krzywe hipereliptyczne w kryptografii” Temat: ECQV

Daniel Trędewicz

Kwiecień 2018

1 Wstęp

Elliptic Curve Qu-Vanstone Implicit Certificate Scheme, czyli tytułowe ECQV, jest schematem niejawnego certyfikowania klucza publicznego [1]. Operacje dokonują się w arytmetyce krzywych eliptycznych, zaś bezpieczeństwo oparte jest na trudności problemu logarytmu dyskretnego.

2 Schemat ECQV

Działanie schematu opiera się na współpracy dwóch stron: centrum akredytacyjnego (certyfikacyjnego) CA oraz podmiotu U chcącego uzyskać certyfikat. Schemat składa się z pięciu ogólnych kroków:

ECQV_Setup CA ustala parametry krzywej eliptycznej, funkcję skrótu, format kodowania certyfikatu oraz generator liczb losowych. Generuje parę klucz prywatny - klucz publiczny (d_{CA}, Q_{CA}) . Wszystkie powyższe dane (oczywiście oprócz d_{CA}) są opublikowane i powszechnie dostępne.

Cert_Request U wysyła żądanie uzyskania certyfikatu do CA składające się z wartości R_U wygenerowanej adekwatną procedurą.

Cert_Generate Po otrzymaniu żądania od U , CA wytwarza certyfikat $Cert_U$ i przesyła go U .

Cert_PK_Extraction Korzystając z certyfikatu $Cert_U$ oraz klucza publicznego CA wylicza się klucz publiczny Q_U podmiotu U .

Cert_Reception Po otrzymaniu odpowiedzi na swoje żądanie U weryfikuje otrzymane dane.

2.1 ECQV_Setup

- CA ustala zbiór parametrów krzywej eliptycznej $\{q, a, b, G, n, h\}$ oraz wielomian nierozkładalny jeżeli $q = 2^m$. Może wygenerować je samodzielnie używając zaaprobowanych metod opisanych w [2], bądź skorzystać z konkretnej rekomendowanej krzywej [3].
- CA wybiera zaaprobowaną funkcję skrótu. Niech H oznacza wybraną funkcję, a $hashlen$ długość wyjścia z niej. Jako H_n oznaczamy wyjście z funkcji interpretowane jako liczba całkowita modulo n .
- CA wybiera zaaprobowany generator liczb losowych (pseudolosowych), wykorzystywany do generowania prywatnych kluczy.
- CA uzyskuje parę kluczy (d_{CA}, Q_{CA}) , gdzie d_{CA} jest liczbą całkowitą uzyskaną z generatora liczb losowych i stanowi klucz prywatny CA , zaś Q_{CA} jest d_{CA} -tą wielokrotnością generatora G grupy punktów na krzywej i stanowi klucz publiczny CA .
- Zbiór parametrów krzywej, wybrana funkcja skrótu, generator oraz Q_{CA} są wszystkie upublicznione. Wszystkie podmioty są w stanie uwierzytelnić powyższe wartości oraz fakt posiadania przez CA klucza prywatnego metodami opisanymi w [2].

2.2 Cert_Request

U uzyskuje parę (k_U, R_U) , gdzie k_U jest liczbą całkowitą uzyskaną z generatora liczb losowych, zaś $R_U = [k_U]G$. R_U jest żądaniem wysyłanym przez U do CA .

2.3 Cert_Generate

- Po otrzymaniu i uwierzytelnieniu wartości R_U , CA generuje liczbę całkowitą k i oblicza $P_U = R_U + [k]G$.
- Wykorzystując jakiś sposób kodowania, CA tworzy certyfikat $Cert_U$ kodując wartość P_U .
- CA oblicza $e = H_n(Cert_U)$
- Jeśli $[e]P_U + Q_{CA} = \infty$, gdzie ∞ jest punktem w nieskończoności, należy wygenerować inne k i powtórzyć procedurę.
- CA oblicza wartość $r = ek + d_{CA}(mod\ n)$ i wysyła w odpowiedzi do U parę $(r, Cert_U)$.

2.4 Cert_PK_Extraction

- Po otrzymaniu odpowiedzi od CA , U odkodowuje wartość P_U z $Cert_U$ i uwierzytelnia ją.
- U oblicza $e = H_n(Cert_U)$
- U oblicza $Q_U = [e]P_U + Q_{CA}$

2.5 Cert_Reception

U oblicza $d_U = r + ek_U(mod\ n)$ i wyznacza $Q'_U = [d_U]G$ i jeżeli $Q_U = Q'_U$ oznacza to, że cały protokół przeprowadzony został pomyślnie. (d_U, Q_U) stanowi parę klucz prywatny - klucz publiczny podmiotu U .

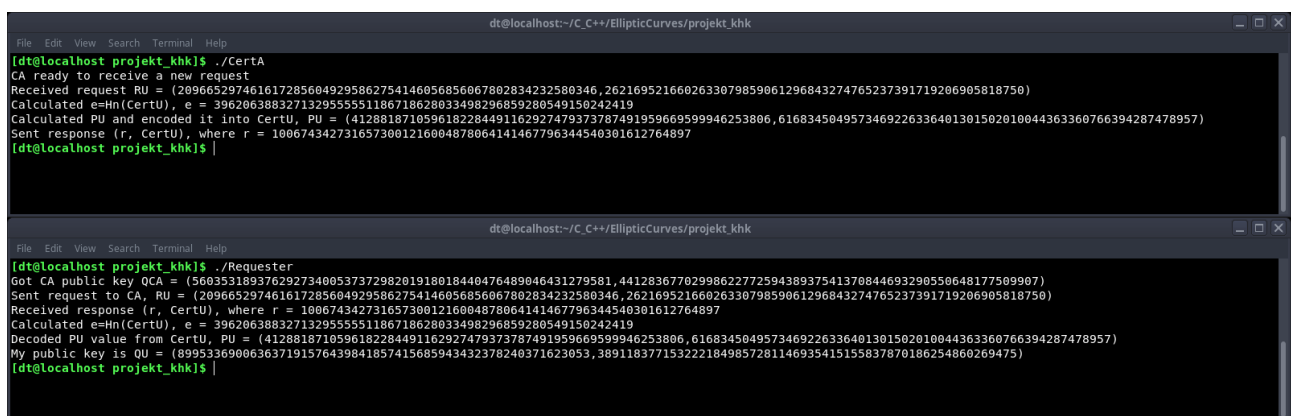
3 Implementacja

Na projekt składają się 3 aplikacje: SetupCA, CertA i Requester. SetupCA realizuje krok ECQV_Setup: ustala krzywą eliptyczną na podstawie podanego argumentu (dostępne tylko krzywe nad ciałem F_p podane w [3]) oraz generuje parę (d_{CA}, Q_{CA}) i zapisuje wszystko do odpowiednich plików. CertA po uruchomieniu czeka na żądanie od Requester'a. Requester realizuje krok Cert_Request, i czeka na odpowiedź z CertA. CertA realizuje krok Cert_Generate. Następnie Requester wykonuje kroki Cert_PK_Extraction oraz Cert_Reception.

Komunikacja i wymiana danych pomiędzy CertA i Requester'em odbywa się za pomocą mechanizmu *named pipes*, zwanym również *FIFO* w systemie Unix-owym (w tym wypadku GNU/Linux).

Aplikacje zostały napisane w języku C++ z wykorzystaniem bibliotek: NTL (dla arytmetyki dużych liczb całkowitych oraz funkcji generowania liczb pseudolosowych) [4], gcrypt (dla funkcji skrótu) [5] oraz ecdt - biblioteki obsługującej arytmetykę na krzywej eliptycznej nad ciałem prostym, mojego autorstwa, napisanej w ramach laboratorium. Kod został skompilowany przy użyciu GCC-C++ v.7.3.1 pod systemem operacyjnym Fedora 27 [6]. Kod źródłowy aplikacji oraz plików, z których złożono bibliotekę ecdt dostępny na [7].

4 Wyniki działania aplikacji



```
dt@localhost:~/C_++/EllipticCurves/projekt_khk
[dt@localhost projekt_khk]$ ./CertA
CA ready to receive a new request
Received request RU = (2096652974616172856049295862754146056856067802834232580346, 2621695216602633079859061296843274765237391719206905818750)
Calculated e=Hn(CertU), e = 3962063883271329555551186718628033498296859280549150242419
Calculated PU and encoded it into CertU, PU = (4128818710596182284491162927479373787491959669599946253806, 6168345049573469226336401301502010044363360766394287478957)
Sent response (r, CertU), where r = 1006743427316573001216004878064141467796344540301612764897
[dt@localhost projekt_khk]$

dt@localhost:~/C_++/EllipticCurves/projekt_khk
[dt@localhost projekt_khk]$ ./Requester
Got CA public key QCA = (5603531893762927340053737298201918018440476489046431279581, 44128367702998622772594389375413708446932905506481775099907)
Sent request to CA, RU = (2096652974616172856049295862754146056856067802834232580346, 2621695216602633079859061296843274765237391719206905818750)
Received response (r, CertU), where r = 1006743427316573001216004878064141467796344540301612764897
Calculated e=Hn(CertU), e = 3962063883271329555551186718628033498296859280549150242419
Decoded PU value from CertU, PU = (4128818710596182284491162927479373787491959669599946253806, 6168345049573469226336401301502010044363360766394287478957)
My public key is QU = (899533690063637191576439841857415685943432378240371623053, 3891183771532221849857281146935415155837870186254860269475)
[dt@localhost projekt_khk]$
```

Przedstawione wyniki uzyskano przy użyciu standardowej krzywej P-192 z [3]. Jak widać, Requester pobiera klucz publiczny CertA i wysyła żądanie, które CertA prawidłowo odczytuje. Następnie CertA dokonuje wszystkie konieczne operacje i przesyła odpowiedź, którą Requester również odczytuje prawidłowo. Na koniec Requester oblicza swoją parę kluczy.

5 Podsumowanie

Zgodne wyniki tak jak powyżej otrzymano dla wszystkich standardowych krzywych nad F_p , pozwala to wysnuć wniosek, że aplikacje, jak również biblioteka `ecdt` oraz komunikacja procesów poprzez mechanizm *named pipe*, zostały prawidłowo zaimplementowane.

Literatura

- [1] SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV). <http://www.secg.org/sec4-1.0.pdf>.
- [2] SEC 1: Elliptic Curve Cryptography. <http://www.secg.org/sec1-v2.pdf>.
- [3] FIPS PUB 186-4, APPENDIX D: Recommended Elliptic Curves for Federal Government Use. <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf>.
- [4] <http://shoup.net/ntl/>.
- [5] <https://www.gnupg.org/software/libgpcrypt/index.html>.
- [6] <https://getfedora.org/>.
- [7] <https://github.com/GaloisField94/ECQV-studies>.