Politechnika Warszawska

Wydział Fizyki

Kryptografia i bezpieczeństwo informacji dla fizyków

Sprawozdanie z zadania nr. 5 na temat:

**Kryptografia krzywych eliptycznych**

Data wykonania zadania w laboratorium: 19.11.2019r.

**Wykonali:**

Maciej Czarnecki

Denys Morokov

Fizyka techniczna II stopień, 2 rok

1. **Wstęp**
   1. **Cel:** celem zadania jest napisanie prostego programu szyfrującego zawartość przy użyciu krzywych eliptycznych z użyciem wybranej biblioteki kryptograficznej. Program ma umożliwiać ustawienie własnych parametrów dla EC. Należy ustawić parametry inne niż domyślne dla biblioteki ale prawidłowe dla EC.
   2. **Wykorzystana technologia:** skrypt został zaimplementowany w Python.
2. **Opracowanie wyników**

Podczas wykonania zadania znaleziono w literaturze i opracowano kilka skryptów do kryptografii na krzywych eliptycznych.

* 1. **Krótki wstęp o krzywych eliptycznych**

Krzywe eliptyczne – to zbiór punktów opisanych równaniem



gdzie



warunek konieczny, aby wykluczyć specjalne krzywe.

**Parametry zakresu**

Algorytmy krzywej eliptycznej będą działać w cyklicznej podgrupie krzywej eliptycznej na polu skończonym. Dlatego algorytmy będą wymagały następujących parametrów:

1. Proste ***p*** określenie rozmiaru końcowego pola.
2. Współczynniki ***a*** i ***b*** równania krzywej eliptycznej.
3. Punkt bazowy ***G*** generujący podgrupę.
4. Kolejność ***n*** podgrup.
5. Kofaktor ***h*** podgrupy.

**Kryptografia na krzywych eliptycznych**

* Klucz prywatny – jest losową całkowitą liczbą ***d*** z zakresu {1, …., n -1}, gdzie n – rząd podgrupy.
* Klucz publiczny – jest punktem ***H = dG*** (G – punkt bazowy podgrupy, „generator”)

Jeśli znamy ***d*** *i* ***G*** (wraz z innymi parametrami krzywej), to znalezienie H jest „łatwe”. Ale jeśli znamy ***H*** *i* ***G***, to znalezienie klucza prywatnego jest „trudnym” zadaniem, ponieważ wymaga rozwiązania dyskretnego logarytmu.

**ECDH (Elliptic curve Diffie-Hellman)**

Ten algorytm jest raczej ***protokołem uzgadniania kluczy*** (protokół kryptograficzny, dzięki któremu dwie (lub więcej) stron może uzgodnić klucz w taki sposób, że obie mają wpływ na rezultat). Zasadniczo oznacza to, że ECDH określa (do pewnego stopnia) kolejność generowania i wymiany kluczy. Możemy sami wybrać metodę szyfrowania danych przy użyciu takich kluczy.

Działa następująco: mamy dwie strony, Alicja i Bob, którzy wymieniają między sobą informację w ten sposób, że trzecia osoba może przechwycić tę informację, ale nie może odszyfrować.

**1.** Najpierw Alicja i Bob generują własne prywatne i publiczne klucze.

Klucze Alicji: **da** – prywatny, **Ha = daG** – publiczny.

Klucze Boba: **db** – prywatny, **Hd = dbG** – publiczny.

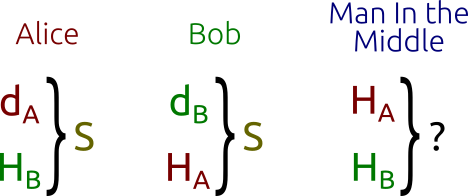
Ale, wykorzystują te same parametry zakresu wspominane wcześniej.

**2.** Wymieniają między sobą klucze publiczne Ha i Hb w sposób niezabezpieczony.

Trzecia osoba (Man In The Middle) przechwytuje Ha i Hb, ale nie jest w stanie określić da i db, ponieważ musi rozwiązać problem logarytmu dyskretnego.

**3.** Alicja oblicza **S = daHb** (wykorzystując własny klucz prywatny i publiczny Boba), a Bob oblicza **S = dbHa**. S jest ten sam dla Boba i Alicji, ponieważ:





Po otrzymaniu wspólnego tajnego klucza Alice i Bob mogą wymieniać dane za pomocą szyfrowania symetrycznego.

**ECDSA (Elliptic Curve Digital Signature Algorithm)**

Algorytm jest wykorzystywany do podpisu cyfrowego.

Alicja chce podpisać wiadomość za pomocą swojego klucza prywatnego (**da**), a Bob chce zweryfikować podpis za pomocą klucza publicznego Alicji (**Ha**). Nikt oprócz Alicji nie powinien być w stanie tworzyć prawidłowych podpisów. Każdy powinien mieć możliwość weryfikacji podpisów. Alicja i Bob wykorzystują te same parametry zakresu.

ECDSA działa z haszem wiadomości, a nie z samą wiadomością. Wybór funkcji skrótu pozostaje po naszej stronie, ale oczywiście musimy wybrać funkcję skrótu kryptograficznego. Skrót wiadomości musi zostać obcięty, aby długość skrótu była taka sama, jak długość bitu **n** (rząd podgrupy). Skrócony skrót jest liczbą całkowitą i będzie oznaczony jako **z**.

Algorytm podpisu wiadomości/dokumentu przez Alicję jest następujący:

1. Wybieram losową liczbę **k** z zakresu {1,….,n – 1}.
2. Obliczamy punkt **P = kG**.
3. Obliczamy liczbę **r = xp mod n**. Jeżeli r = 0, to wybieramy inną liczbę k i powtarzamy kroki.
4. Obliczamy **s = k-1(z+rda) mod n**. Jeżeli s = 0, to wybieramy inna liczbę k i powtarzamy kroki.

**(r, s)** – jest podpisem.

Czyli, Alicja podpisuje hasz (dokument) **z** za pomocą prywatnego klucza **da** i losowego **k**. Bob weryfikuje podpis wiadomości za pomocą klucza publicznego Alicji **Ha**.

Działa dany algorytm jedynie, jeżeli **n** jest liczbą pierwszą.

**Dlaczego EC są bezpieczniejsze od innych szyfrów kryptograficznych**

**Problem logarytmu dyskretnego.**

Jeśli znamy **P i Q**, to ile powinno wynosić **k**, żeby **Q = kP**. ECC jest interesujące pod tym względem, że obecnie problem dyskretnego logarytmu dla krzywych eliptycznych wydaje się „bardziej skomplikowany” w porównaniu z innymi podobnymi zadaniami stosowanymi w kryptografii. Oznacza to, że potrzebujemy mniej bitów dla **k**, aby całość mogła uzyskać taki sam poziom ochrony, jak w innych kryptosystemach.

| **Rozmiar klucza RSA [bit]** | **Rozmiar klucza EC [bit]** |
| --- | --- |
| 1024 | 160 |
| 2048 | 224 |
| 3072 | 256 |
| 7680 | 384 |
| 15360 | 521 |

Dobrze to pokazuje tabela poniżej, która porównuje rozmiar klucza RSA z rozmiarem klucza EC (w bitach), tabela udostępniona przez NIST.

Tabela 1. Porównanie rozmiarów kluczy.

Jak widać, nie ma liniowej zależności między rozmiarem klucza RSA a kluczem EC (innymi słowy: jeśli podwoimy rozmiar klucza RSA, nie będziemy musieli podwajać rozmiaru klucza EC). Tabela mówi nam, że EC nie tylko zużywa mniej pamięci, ale także generowanie kluczy z logowaniem w niej jest znacznie szybsze.

* 1. **Uzyskane wyniki.**

**zad5\_own\_parameters\_ec.py, parameters.xml** – stworzony skrypt do generacji kluczy publiczny i prywatnych z możliwością ustawienia własnych parametrów dla krzywej eliptycznej. Wykorzystano bibliotekę *fastecdsa*, ale niestety zawiera błąd związany z zamianą typu wprowadzonych parametrów do wymaganych przez funkcję tworzącą krzywą eliptyczną.

**zad5\_cryptography\_signature.py, settings.xml** – skrypt, który wykorzystuje bibliotekę *cryptography* do tworzenia krzywej eliptycznej o określonej nazwie (nazwa jest podawana w settings.xml), następnie podpisuje wiadomość.

**zad5\_ecdh\_literature.py** – skrypt pobrany ze strony źródła [5] do realizacji algorytmu ECDH opisanego powyżej.

**zad5\_ecdsa\_literature.py** - skrypt pobrany ze strony źródła [5] do realizacji algorytmu podpisu cyfrowego ECDSA opisanego powyżej.

1. **Podsumowanie**

Zaimplementowano skrypt do podpisu wiadomości dla wybranej EC, opisano w skrócie zasadę działania kryptografii krzywych eliptycznych (algorytmy ECDH i ECDSA).

1. **Źródła**

1. <https://pl.wikipedia.org/wiki/Kryptografia_krzywych_eliptycznych>

2. <https://pl.wikipedia.org/wiki/Protok%C3%B3%C5%82_uzgadniania_kluczy>

3. <https://habr.com/ru/post/335906/>

4. <https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/>

5. <https://github.com/andreacorbellini/ecc>