Prof. Dr.-Ing. Christof Paar
M. Sc. Paul Staat
M. Sc. Simon Klix
B. Sc. René Walendy

Pfingstprojekt SoSe23

Abgabe: 26.06.2023 **bis** 11:45 Uhr

Dieses Pfingstprojekt behandelt Anwendungs- und Implementierungsaspekte der asymmetrischen Kryptographie. Ziel ist es, den Umgang und die Implementierung verschiedener Algorithmen zu vertiefen.

Abgabemodalitäten und Bewertungskriterien

Das freiwillige Projekt bringt bis zu 100 Punkte zusätzlich zu den Hausaufgaben. Mit diesen Punkten können in den Hausaufgaben verlorene Punkte ausgeglichen werden. Die Bearbeitung des Projekts ist nicht verpflichtend. Auch ohne die Bearbeitung des Projekts ist es theoretisch möglich, alle Bonuspunkte zu erhalten, sofern alle Hausaufgaben vollständig korrekt abgegeben wurden. Auch erhöht sich durch die Bearbeitung des Projekts nicht die Anzahl der maximal erreichbaren Bonuspunkte.
Das Pfingstprojekt wird in denselben Gruppen bearbeitet und abgegeben wie die Hausaufgaben. Für die volle Punktzahl müssen alle Aufgaben bearbeitet werden und Ihr Code ohne Warnungen und Fehlermeldungen ausführbar sein. Für Abgaben, die diese Kriterien nicht erfüllen, werden Teilpunkte vergeben. Abschreiben führt dazu, dass das gesamte Projekte mit 0 Punkten bewertet wird. Wir weisen darauf hin, dass wir bei der Korrektur solcher Programmierprojekte standardmäßig Software zur Plagiatsprüfung einsetzen. Sie können sich gegenseitig beim Verständnis der Algorithmen helfen oder grundsätzliche Tipps zur Programmierung geben, solltet es aber dringend vermeiden, Programmcode untereinander (also außerhalb eurer Abgabegruppen) auszutauschen.
Wir stellen Code-Templates in Python zur Bearbeitung der Programmieraufgaben bereit. Bitte nutzen Sie für Ihre Abgaben *ausschließlich* diese Templates. Achten Sie in Ihrem Quellcode stets auf sinnvolle **Kommentare**!
Die Abgabe erfolgt in zwei einzelnen Dateien (Antworten zu Aufgaben als PDF, Python-Code als Textdatei).

Entwicklungsumgebung

Als Programmiersprache für die Implementierung soll ausschließlich `Python3` verwendet werden. Die Verwendung der entsprechend bereitgestellten Vorlagen ist verpflichtend, da die Korrektur bei der erwarteten Anzahl an Abgaben für uns ansonsten nicht zu bewerkstelligen ist. Die Vorlagen enthalten bereits die grobe Struktur der resultierenden Programme und geben (hoffentlich) hilfreiche Hinweise zur Implementierung der einzelnen Komponenten. Außerdem ist für die Programmierung in `Python3` (mindestens) die Version `3.7` vorgeschrieben. Beachten Sie bitte, das wir keinen Support zu Entwicklungsumgebungen geben können. Wir empfehlen die Nutzung von *Visual Studio Code*[1] oder *PyCharm*[2], es kann allerdings auch jeder beliebige andere Texteditor verwendet werden.

[1] `https://code.visualstudio.com`
[2] Kostenlose Version oder Bildungslizenz, `https://www.jetbrains.com/community/education/\#students`

Einleitung

In diesem Projekt implementieren wir den Diffie-Hellman Schlüsselaustausch auf Basis von elliptischen Kurven über $GF(p)$. Wir betrachten Optimierungsmöglichkeiten und testen die Leistungsfähigkeit unserer Implementierung.
Gegeben sei die folgende elliptische Kurve:

$$\mathcal{E} : y^2 = x^3 + ax + b \bmod p$$

und der Basispunkt $P = (x, y)$ mit der Ordnung $q = ord(P)$ (in hexadezimaler Schreibweise):

$$\begin{aligned}
p &= \texttt{0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff} \\
a &= \texttt{0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc} \\
b &= \texttt{0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b} \\
x &= \texttt{0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296} \\
y &= \texttt{0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5} \\
q &= \texttt{0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551}
\end{aligned}$$

Die gegebenen Parameter entsprechen der Kurve P-256[3] aus dem offiziellen Standard NIST SP 800-186[4]. Es handelt sich um eine der Kurven, die seitens der NIST für US-Regierungsanwendungen zugelassen sind.

1. Schnelle Punktmultiplikation: Non-Adjacent Form (NAF) 50 Punkte

Aus der Vorlesung *Einführung in die Kryptographie 1* kennen Sie bereits den Square-and-Multiply Algorithmus zur schnellen modularen Exponentiation – ein wichtiger Algorithmus, der bspw. für RSA und DH benötigt wird. Darauf aufbauend wurde in *Einführung in die Kryptographie 2* der Double-and-Add Algorithmus zur schnellen Punktmultiplikation auf elliptischen Kurven eingeführt. Sie haben sicher bereits gehört, dass beide Algorithmen noch weiter optimiert werden können, sodass noch weniger Operationen (sprich: Zeit) benötigt werden. Dazu verwenden wir in diesem Pfingstprojekt die sog. *Non-Adjacent Form* (NAF).

a) Eine erste Einführung in das Thema finden Sie in der EK2 Globalübung 18 aus dem Jahr 2022, deren Aufzeichnung unter folgendem Link abrufbar ist: `https://youtu.be/PAOTEusXBmE`. Lesen Sie zur weiteren Vertiefung den über Moodle zur Verfügung gestellten Auszug aus dem Skript zur Vorlesung *Implementierung kryptograpischer Verfahren* (`auszug_IKV1.pdf`).

Wir beginnen nun schrittweise mit der Python-Implementierung. Verwenden Sie dazu **ausschließlich** das bereitgestellte Python-Template `ecc_template.py` und die darin enthaltenen Implementierungen für die Berechnung von Gruppenoperationen auf elliptischen Kurven.
Hinweis: In Python können überladene Operatoren verwendet werden. Nutzen Sie zur Überprüfung Ihrer Implementierung das Skript (`testbench.py`).

[3] `https://neuromancer.sk/std/nist/P-256/`
[4] `https://csrc.nist.gov/publications/detail/sp/800-186/final`

b) Implementieren Sie zunächst den normalen Double-and-Add Algorithmus (wie aus der Vorlesung bekannt) in der Funktion `double_and_add`. (10 Pkt.)

c) Implementieren Sie die Berechnung der NAF aus Algorithmus 0.1 des IKV-Skripts in der Funktion `calc_naf_representation`. (20 Pkt.)

d) Implementieren Sie einen angepassten Double-and-Add Algorithmus, der die Punktmultiplikation basierend auf der NAF ausführt (Algorithmus 0.2 aus dem IKV-Skript). Verwenden Sie dazu die Funktion `calc_naf_representation`. Berechnen Sie $P = k \cdot Q$ für die folgenden Werte und geben Sie das Ergebnis in Ihrer Abgabe an. Nutzen Sie für die Berechnung die oben bereits gegebene elliptische Kurve P-256. (10 Pkt.)

Q ={`0xb0a3d57e543778709f9d74910deb1b5b2c6405c87fff60a175a68e866b0388b3`,
`0xcf2879fd7eed0ac3000c484c488a6256c3887f39afdd62e764a74e814c7e7ebe`}
k =`0xFEEDDEADBEEFBADCAB1EDECAFBADC0DEC001D00DC0DEBA5EC0CAC01AADD511FE`

e) Fügen Sie Zählvariablen ein, um auszugeben, wie viele Operationen (Double, Add, Sub) benötigt werden, um $P = k \cdot Q$ zu berechnen: (10 Pkt.)

(i) für den Double-and-Add Algorithmus (Teilaufgabe 1.b),
(ii) für den angepassten Double-and-Add Algorithmus mit NAF (Teilaufgabe 1.d).

Geben Sie das Ergebnis in Ihrer Abgabe an. Welcher der beiden Algorithmen ist vorteilhafter? Begründen Sie in *maximal 3 Sätzen* weshalb.
Hinweis: Sie können die entsprechenden `print()`-Aufrufe in den folgenden Teilaufgaben auskommentieren, falls die Ausgabe dort störend ist.

2. Diffie-Hellman-Schlüsselaustausch mit Elliptischen Kurven (ECDH)
50 Punkte

Wir betrachten nun einen Diffie-Hellman-Schlüsselaustausch auf Basis der zuvor gegebenen elliptischen Kurven (ECDH) mit 256-Bit-Parametern. Alice und Bob nutzen den ECDH, um den gemeinsamen Schlüssel T_{AB} zu etablieren.
Fügen Sie Ihre Implementierungen zu den folgenden Teilaufgaben an den jeweils gekennzeichneten Stellen in das Python-Template `ecc_template.py` ein.

a) Berechnen Sie unter Zuhilfenahme der Funktion aus Aufgabe 1.d den gemeinsamen ECDH-Schlüssel K_{AB} zu den folgenden privaten ECDH-Parametern.
$k_{priv,A}$ = `0x30b62acc9178ca5a5099651318404f16e46ede2be314cbc754a96fa91292433d`
$k_{priv,B}$ = `0xf1074ff845239e327797565d9f253180c11044beeadf90e620863e97a6b24040`
(10 Pkt.)

b) Testen Sie nun die Geschwindigkeit Ihrer Implementierung. Messen Sie die benötigte Gesamtzeit, um die Berechnungen aus Aufgabe 2.a 1.000 mal auszuführen. Führen Sie diese Messung für die beiden implementierten Verfahren zur Punktmultiplikation aus den Aufgaben 1.b und 1.d durch. Verwenden Sie dazu die Funktionen `double_and_add` und `naf_double_and_add`. Geben Sie die beiden gemessenen Geschwindigkeiten an und erläutern Sie Ihr Ergebnis.

(15 Pkt.)

c) Passen Sie die Zeitmessung aus Aufgabe 2.b an, sodass Sie nun die Dauer einzelner Aufrufe der Funktionen `double_and_add` und `naf_double_and_add` messen. Rufen Sie die Funktionen jeweils 5.000 mal auf, um entsprechend 5.000 einzelne Zeitmessungen für jeden der Algorithmen aufzuzeichnen. Nutzen Sie die bereitgestellte Funktion `plot_performance` um die gemessenen Zeiten als Histogramme zu visualisieren. Die erzeugte Grafik wird als PNG-Datei abgespeichert. Fügen Sie diese in Ihre PDF-Abgabe ein.
Hinweis: Für diese Teilaufgabe muss das Python-Paket `matplotlib`[5] installiert sein.

Diskutieren bzw. beantworten Sie knapp die folgenden Punkte:

(*i*) Beschreiben Sie die Grafik und benennen Sie die Haupterkenntnisse.
(*ii*) Weshalb ist es sinnvoll, die Geschwindigkeitsmessungen sehr oft zu wiederholen?
(*iii*) Sind die beobachteten Geschwindigkeitsunterschiede generalisierbar (bspw. auf andere Computer oder für andere Eingabewerte)?

(20 Pkt.)

Der gemeinsame Schlüssel T_{AB} aus dem ECDH wird zur Dateiverschlüsselung verwendet, die mit dem symmetrischen Verschlüsselungsverfahren AES-128 erfolgt. Für die Ableitung des AES-Schlüssel wird ausschließlich die x-Koordinate des Punktes T_{AB} verwendet. Da allerdings $K_{AB} = x_{T_{AB}}$ direkt 256 Bit liefert, jedoch für die AES-Verschlüsselung nur 128 Bit benötigt wird, kommt eine Reduktionsfunktion $\xi : \{0,1\}^{256} \mapsto \{0,1\}^{128}$ zum Einsatz (fehlende Bitstellen werden mit 0 aufgefüllt):

$$\begin{aligned} X &= (x_{255}, \ldots, x_{128}, x_{127}, \ldots, x_0)_2 \\ \xi(X) &= (x_{255}, \ldots, x_{128})_2 \oplus (x_{127}, \ldots, x_0)_2 \end{aligned}$$

d) Implementieren Sie die Funktion ξ, um den Schlüssel $K_{AES\,128}$ aus T_{AB} zu erzeugen. Entschlüsseln Sie mit dem Schlüssel $K_{AES\,128} = \xi(T_{AB})$ die Datei `ciphertext.hex` im Projektordner mittels AES128-ECB. Geben Sie in Ihrer Lösung an, um welchen Dateityp es sich hier handelt.
Hinweis: Sie können für die Entschlüsselung das Programm CrypTool[6] (Ver-/Entschlüsseln → Symmetrisch (modern) → Rijndael (AES)...) oder CyberChef[7] (AES Decrypt) verwenden.

(15 Pkt.)

[5] `https://matplotlib.org/stable/users/getting_started/index.html`
[6] `https://www.cryptool.org/de/ct1-downloads`
[7] `https://gchq.github.io/CyberChef/`