Asymmetric Cryptanalysis & MPC – Submission Guidelines

- Timeline: Release **19. 03. 2020**; Question time **23. 04. 2020**; Submission **30. 04. 2020**
- Upload your team's submission on `https://stics.iaik.tugraz.at/`. Each task is uploaded separately. Don't forget to tick the selected tasks!
- Submit your code as a {`zip`, `tar.gz`} archive. Add a file `README.`{`md`, `txt`, `pdf`} on the top level that documents your submission (design choices, limitations, howto, runtime).
- You can use your favourite programming language and libraries for the necessary big-integer and modular arithmetic. Consider using a computer algebra system, such as SAGE (`https://sage.tugraz.at` with TUGRAZonline login). Please document your choices, as well as how to compile and use your implementations in the `README`. If you intend to use non-free software, please clarify with us beforehand.

1–A Wiener's Attack on RSA (4 Points)

Implement the attack by Wiener [Wie90] to recover small RSA private exponents d.

(a) Continued fractions for $\mathbb{Q}$ (2 Points): Implement the computation of n-th convergents of the continued fraction expansion to approximate rational numbers.

(b) Recover RSA private key (2 Points): Use your implementation to recover Bob's private key d, p, q from the following 4096-bit RSA public key in PEM format:

```
-----BEGIN PUBLIC KEY-----
MIIEIjANBgkqhkiG9w0BAQEFAAOCBA8AMIIECgKCAgEAqpEukmN0/51bylLv5iY9
LiAxSwTb6WOIN/+3vhueUcjmjWhB+XBtT9zsvRpD1yRvWAlnmtaOrwKKd9QNrv9S
4nSdXwCRWYLkthWGfypMCT2cfz2M5/kQDJr8ed+Kqb/ZXwReIENcA5H76JaWSwqy
ZPx83Ud9vKC0ApQ+Y9aS0xiNCGqlMf2o6h7RBL1a+SeL232zhGaCw5pg1sfHTh4Z
gZLnPhfzJX0XcZMMcJ5KeIUuFCa3MFt+RH1+UPjDQ8Kz7ULYBIJyBTh/B2CN8oo7
9TosF+WgPOTP5tpx11lB+GzHmSJscQ4G6plBNCw+QeyfS2XRSkgnfm7dh9LMr6a+
s6Hlf3/BEvXFa67SG7nw9RPbmO65pnN02NNRh2/oXpesJTFzUNvMaxNEy6geRaVb
LbiiPjUneo2x5QmFYqPvuYuJUeQhyXMOGJRbtyw0AtVcRc2qA5PYehG2FO3OHo3B
QvfwR1U32cpkHd1bPigttdsmZ2cpk39H9R9f+drHTr83chk0/A7m6pJIAp8aftSQ
W5x1L7qvjIrt1iQMjJOir7MOaznr1+/rjIVLkSZMbzBiKI2pcgXOh3j7euV708tf
XFWJhBt2cCUNBJywRji6B5Zt9/HvkKv50qxto43Vz9RBA/JlcKszIU87QBEi7wYr
LGQxIZUmmUiC9qwJPgHPlJECggIBAJ0QcXSCwPD6OuLy9+U6TthE2k57zovR+1V7
bjNJ7w9wo5o1Fc1RioNDomVwnOuPxVJV+aAZD8MofWtHMfNdrtAdH8JzS9AqR1E5
+qK/em5e60db/fcGXRDG09AW4VkIxTtl65tArOqZHSrYikCyfInGFNUjNvDBmw10
BApN7VHcJQnI70wuwWsLGp9cLUN1u6AjH7rPXM+NZJ0UliA2nupAQpNz4kbBhDR7
viMw3vuC1XjFWVhi4WoXmMbwV2XNHo83SZb2Qd9/JSb8qRbfsHs839cAq1oLiZow
x1Ita5GbPelY3yrMjEmPmNIOO5iiCwUeI/U1L+w5b/DVEWxVUSRkV5FB2UOmD9lY
VkpqTebACYeQBtFFOhfEVkBzFfjg7xKaPaks9w3xGX1AIcBCfeY7FL4MT+NOroDw
H+F5mZIgIfn/pbxaC0BaUPJKRGAYRBY+Y3/tBX4R4jv1nWEk+GIsGoJl6rz1M333
ZItOqqTIDWHyk/EJD4t/yRe+ziekOsFo6XoY8e8GMdhDeHBRK5weM3AIoLEURtK8
UAcrHC1DoAWAcaC/GspvTtHVnB5uxHWpX5cMuAelmkMVBqASY5lg70e1Ei1t2XD+
zQsaOGOvMA9V7C7/QsYv3Isg/NI7vRY9DyhC5p0jJWf6/VhGeiNt3v9QnzKCIXu5
FN7A/Vpp
-----END PUBLIC KEY-----
```

Use the private key to decrypt the following base64-encoded ciphertext encrypted using

RSA-PKCS1-v1_5.

```
jc+dvDNNgj0ifVEojoisDi8PBqVUYQe4mTZahlAuIJvh3+aXAoN8VWUYIqnBmn3QWrXQ2btqcA5Z
8COCEKNONZuTV2bMlTcywZW/9nDq2cdEt/7ej4eNRG9APVGAAWUUM1soofpB0qq/y94pzU3lgoTs
koQ61GWfVV3s8ztJhCzs3XFqJ64pS+3MgvuMuaP90xAlnm9HaZttTYugigYEvl6c6lJRELeV7SIZ
j9WDZFfEoKphFmC62Zm0UpJJ0RYGQ9r2sA283mWc/RptYAuVn9gJWyP+BgZP0e2Zyt0ArbSyPu/s
QywFNKXvuwKmMXJFqSuI3p86MrjR6zFm7fmaG/ezPPyxVdGPsSbfD8PPA0Qh/SwkyfH7VxeZjrKt
ZjZf5U+KLjGJd9Ap0igIabt0X0EhQ0DQqI/9m6Wr/rRyp/l1h+LorMZAVXr7wPokPNmDZ88pmWLH
OsAt/6OHnZ1MLPZPfs23ngMcuPRchaJ0oP1XtPqgSq+cRCmLZJf5M2pgsxUlWkw5kUrU+KxhvhSz
mMoO1G81mg8c4U3z6lKfNELqZGTJyzSjXLzeshh7rLIvtsHvBP6a6RZhRdx0PkJ4tFxXG4WGMC6X
2RhIibUsMZgkvztxxcpMhYdsEGymuVCgyNy6KZFA81mYub+d4ae+KEXF3u0YYpckT2V2Gbia+NM=
```

If you create a valid private key file, you can use the following openssl command to do so.

```
base64 -d ciphertext.b64 > ciphertext
openssl rsautl -in ciphertext -out plaintext.txt -inkey private_key.der -keyform DER -decrypt
```

You can also use other cryptographic libraries like `pycrpytodome` to decrypt the ciphertext once you have recovered the private key.

[Wie90] M. J. Wiener. "Cryptanalysis of short RSA secret exponents". In: IEEE Transactions on Information Theory 36.3 (1990), pp. 553–558. DOI: 10.1109/18.54902.

1–B Multiparty Computation with Oblivious Transfers (8 points)

Implement a 1-out-of-2 Oblivious Transfer (OT) protocol (e.g., Naor-Pinkas [NP01]). Use this 1-out-of-2 OT to build a 1-out-of-N OT and subsequently evaluate a small circuit using a GMW-style protocol.

(a) 1-out-of-2 OT (3 Points): Implement a variant of 1-out-of-2 Oblivious Transfer (e.g., Naor-Pinkas). Test the implementation by obliviously receiving one of two strings from a party.

(b) 1-out-of-N OT (2 Points): Implement a variant of 1-out-of-N Oblivious Transfer. You can either use a scheme that can be instantiated as 1-out-of-N directly (e.g., Naor-Pinkas), or use the method detailed in the lecture to build a 1-out-of-N OT from 1-out-of-2 OTs.

(c) Apply 1-out-of-4 OT to build a GMW-style protocol and evaluate a small circuit (3 Points): Using the 1-out-of-N OT implementation from the previous task, implement a GMW-style protocol to securely evaluate boolean gates. Implement the secret-sharing functionality needed for the input and output sharing phase. Evaluate a 4-bit addition circuit where both parties input a 4-bit integer value and get the addition of both values as a 5-bit output. *Hint:* The addition circuit is already implemented in the python skeleton.

(d) (Bonus) Build and evaluate a circuit for an AES-128 encryption (3 Points): Build a circuit performing an AES-128 encryption. Evaluate it with one party supplying the plaintext and the other party supplying the secret key, using the protocol you build in (c). *Hint:* You can find circuit representations of AES-128 online[1], you do not need to come up with this circuit yourself. *Hint:* An optimized AES-128 encryption circuit has about 5120 AND gates. You should try to estimate the time for an evaluation based on the speed of your 1-out-of-4 OT.

You are expected to test your implementations yourself using a variety of inputs. The length of the transmitted strings in the Oblivious Transfers should be at least 128 bits, but you are free to transmit larger strings if it suits your implementation.
Hint: Use existing libraries for any basic cryptographic primitives like AES (e.g., `pycrpytodome` in python). If you want to implement an elliptic curve variant, use an existing EC library (e.g., `fastecdsa` in python).
Hint: We provide a basic python skeleton that handles networking for you. If you prefer other programming languages, you have to implement something similar yourself.

[NP01] M. Naor and B. Pinkas. "Efficient oblivious transfer protocols". In: SODA. ACM/SIAM, 2001, pp. 448–457.

[1] e.g. `https://homes.esat.kuleuven.be/~nsmart/MPC/`

1–C Factoring with Factor Bases and Sieving (12 Points)

Implement the Quadratic Sieve algorithm [Pom85], and use it to factor the RSA modulus N.

(a) Factoring with factor bases (3 Points): Implement factoring with Dixon's method: For all a_i in some interval $[\sqrt{N} - C, \sqrt{N} + C]$, factor $b_i = a_i^2 - N$ with respect to a factor base $\mathcal{B}$ of small primes $\leq B$. Collect the a_i with smooth b_i.

(b) Combining the relations (3 Points): Use linear algebra to combine these factorizations to find solutions x and y such that $x^2 \equiv y^2 \pmod{N}$. Test by factoring some small N.

(c) Sieving speedup (3 Points): Sieve the interval for smooth a_i: For each $p \in \mathcal{B}$, find the 0–2 solutions α such that $\alpha^2 \equiv N \pmod{p}$, then only the values $a_i \equiv \alpha \pmod{p}$ need to be tested for divisibility by p. *Hint:* You can remove any $p \in \mathcal{B}$ with no solution α from $\mathcal{B}$.

(d) Apply to factor 100-bit number (3 Points): Combine your ingredients to factor $N =$ `0x1c9be820f9caaa0d7107085a5`. *Hint:* Choose an interval size $C \leq \exp(\sqrt{\ln N \cdot \ln \ln N})$ and a smoothness bound $B \leq \sqrt{C}$.

[Pom85] C. Pomerance. "The Quadratic Sieve Factoring Algorithm". In: Advances in Cryptology – EUROCRYPT 84. Ed. by T. Beth, N. Cot, and I. Ingemarsson. Vol. 209. LNCS. Springer, 1985, pp. 169–182. DOI: `10.1007/3-540-39757-4_17`.