Cryptography and Security — Midterm Exam
Solution

Serge Vaudenay

26.11.2014

- duration: 1h45
- no documents allowed, except one 2-sided sheet of handwritten notes
- a pocket calculator is allowed
- communication devices are not allowed
- the exam invigilators will **not** answer any technical question during the exam
- readability and style of writing will be part of the grade

The exam grade follows a linear scale in which each question has the same weight.

1 Generating Prime Numbers

The following exercise is inspired from Efficient Generation of Prime Numbers *by Joye, Paillier, and Vaudenay, published in the proceedings of CHES'00 pp. 340–354, LNCS vol. 1965, Springer 2000.*

We recall that if we pick a random number in $\{1, 2, \ldots, N\}$, the probability that it is prime is approximately $\frac{1}{\ln N}$.

We want to generate prime numbers p and q for an RSA modulus with exponent $e = 3$. To generate one ℓ-bit prime number, we iteratively pick a random number between $2^{\ell-1}$ and $2^{\ell} - 1$ until we find a prime number:

```
GenPrime($\ell$)
1: repeat
2:     pick $p \in \{2^{\ell-1}, 2^{\ell-1}+1, \ldots, 2^{\ell}-2, 2^{\ell}-1\}$ at random
3: until $p$ is prime
4: output $p$
```

Then, to generate a 2ℓ-bit RSA key, we proceed as follows:

```
GenRSA($\ell$)
1: repeat
2:     $p$ = GenPrime($\ell$)
3:     $q$ = GenPrime($\ell$)
4: until $e = 3$ is a valid exponent with the RSA modulus $pq$
5: output $p, q$
```

In this exercise, we assume that ℓ is large enough for the RSA security.

Q.1 Estimate the probability that a randomly selected element from $\{2^{\ell-1}, 2^{\ell-1}+1, \ldots, 2^{\ell} - 2, 2^{\ell} - 1\}$ is prime.

The number of prime numbers in this set is $(2^{\ell} - 1)P(2^{\ell} - 1) - 2^{\ell-1}P(2^{\ell-1}) \approx \frac{2^{\ell}-1}{\ell \ln 2} - \frac{2^{\ell-1}}{(\ell-1)\ln 2} \approx \frac{2^{\ell-1}}{\ell \ln 2}$. *So, the probability is approximately* $\frac{1}{\ell \ln 2}$.

Q.2 Show that the GenPrime algorithm can be speeded up by a factor 2 by selecting random elements in $\{2^{\ell-1}+1, 2^{\ell-1}+3, \ldots, 2^{\ell}-3, 2^{\ell}-1\}$.

The complement of this set has no prime numbers. So, all prime numbers are in this set of odd numbers. This means that by picking elements in this set, the probability that it is prime is twice what it was with the complete set. So, the number of trials is divided by two.

Q.3 Show that $e = 3$ is a valid RSA exponent if and only if p and q are equal to 2 modulo 3.

We know that e is valid for the modulus pq if and only if $\mathsf{gcd}(e, \varphi(pq)) = 1$. *This is if and only if* $\mathsf{gcd}(3, (p-1)(q-1)) = 1$. *This is if and only if* $\mathsf{gcd}(3, p-1) = 1$ *and* $\mathsf{gcd}(3, q-1) = 1$.
We still have to show that for a prime p, $\mathsf{gcd}(3, p-1) = 1$ *is equivalent to* $p \bmod 3 = 2$.
Actually, $p \bmod 3$ cannot be 0 (since p is large and prime). So, $p \bmod 3$ can only be 1 or 2. Clearly, the case of 3 being coprime with $p-1$ can only correspond to the $p \bmod 3 = 2$ case.
Hence, 3 is valid for the modulus pq if and only if $p \bmod 3 = 2$ and $q \bmod 3 = 2$.

Q.4 Consider the following algorithm:

```
GenRSA'(ℓ)
1: repeat
2:    $p = \mathsf{GenPrime}(\ell)$
3: until $p \bmod 3 = 2$
4: repeat
5:    $q = \mathsf{GenPrime}(\ell)$
6: until $q \bmod 3 = 2$
7: output $p, q$
```

Show that it produces equivalent outputs to GenRSA but with a twice lower expected complexity.

GenRSA *has to generate some primes p and q such that $p \bmod 3 = 2$ and $q \bmod 3 = 2$, due to the previous question. So, by putting the requirement on the value modulo 3 earlier, we have an algorithm producing equivalent outputs.*
A random prime number is equal to 2 modulo 3 with probability $\frac{1}{2}$. So, in the previous case, we had to iterate 4 times the generation of the two primes to have them both equal to 2 modulo 3. In total, we had to run the prime number generation 8 times.
With the new algorithm, we iterate the prime number generation twice for each of the prime numbers, so have 4 prime number generations in total.
This is twice faster.

Q.5 The previous way to generate prime numbers is equivalent to using the following new algorithm:

```
GenPrime'(ℓ)
1: repeat
2:    pick $p \in \{2^{\ell-1}, 2^{\ell-1}+1, \ldots, 2^{\ell}-2, 2^{\ell}-1\}$ at random
```

```
3: until $p$ is prime and $p \bmod 3 = 2$
4: output $p$
```

Propose another algorithm GenPrime″ (we expect a full description of the algorithm in the same style as GenPrime′) to generate the prime numbers which is about 6 times faster than GenPrime′. Conclude that GenRSA with this new algorithm instead of GenPrime is speeded up by a factor of about 12.
HINT: a Chinese proverb says that if you have two requirements at the same time, maybe you should combine them into a single requirement.

Q.2 suggests to focus on numbers which are equal to 1 modulo 2. Now, we want to focus on numbers which are equal to 2 modulo 3. Based on the Chinese Remainder Theorem, we could just focus on numbers which are equal to 5 modulo 6. We consider the following algorithm:

GenPrime″(ℓ)

```
1: repeat
2:     pick $x$ such that $2^{\ell-1}/6 \le x < 2^\ell/6$ at random
3:     set $p = 6x + 5$
4: until $p$ is prime
5: output $p$
```

Clearly, is produces equivalent outputs but avoids all values modulo 6 which will make p invalid. So, this new algorithm is 6 times faster than GenPrime*.*
When put in GenRSA′*, this is thus 6 times faster. Since* GenRSA′ *is twice faster than* GenRSA*, we obtain an algorithm which is 12 times faster than* GenRSA*.*

Q.6 Propose a way to speed up GenPrime″ by a factor $\frac{4}{5} \times \frac{6}{7} \times \frac{10}{11} \times \frac{12}{13} \times \frac{16}{17} \times \frac{18}{19} \times \frac{22}{23} \times \cdots$

Instead of picking numbers at random, we should make sure that they are not multiples of 5, 7, 11, 13, ... For this, we can pick random numbers in $\mathbf{Z}_5^*$, $\mathbf{Z}_7^*$, $\mathbf{Z}_{11}^*$, $\mathbf{Z}_{13}^*$, ..., combine them with the Chinese Remainder Theorem together with the 5 modulo 6, obtain a suitable residue r modulo $m = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdots$, and take $p = mx + r$ for $x \in [2^{\ell-1}/m, 2^\ell/m[$.
For instance, if we continue until the prime number 23, the complexity is twice faster than the method of the previous question since

$$\frac{4}{5} \times \frac{6}{7} \times \frac{10}{11} \times \frac{12}{13} \times \frac{16}{17} \times \frac{18}{19} \times \frac{22}{23} \approx 0.49$$

2 Encoding Messages in Elliptic Curves

We consider the ElGamal cryptosystem over an elliptic curve. I.e., we work over a field $\mathbf{Z}_p$, use parameters a, b to define the curve $y^2 = x^3 + ax + b$, and use a generator P of the curve, who has a prime order n. (We recall that n is close to p, due to the Hasse Theorem.) Given a secret key d, the public key is $Q = dP$. Normally, we encrypt group elements. To encrypt a point M in the curve, we compute $R = rP$ for $r \in_U \mathbf{Z}_n$ and $S = M + rQ$. The ciphertext is (R, S).

We want to encrypt bitstrings (of fixed length which is less than $\log_2 n$). To encrypt a bitstring m, we map it to a point on the elliptic curve $M = \mathsf{map}(m)$ then encrypt M. We assume that map is efficiently invertible so that after decrypting (R, S) we can apply map^{-1} to obtain m. In this exercise, we consider the problem of defining map.

Q.1 Given the secret d and the parameters (p, a, b, n, P) recall how the above ElGamal cryptosystem is constructed from the semi-static Diffie-Hellman protocol. Then, give the method to decrypt the ciphertext (R, S).

We have $rQ = rdP = dR$. This is actually the Diffie-Hellman property: the receiver selects a long-term secret d and a public key $Q = dP$ and the sender selects an ephemeral secret r and a public $R = rP$. The key on which they agree is rdP, computed as rQ on the sender side and as dR on the receiver side.
The cryptosystem is performing this Diffie-Hellman-like protocol, then encrypt the point M by the generalized Vernam cipher with key rdP, i.e., by adding these two points.
So, $M = S - dR$ which is computed with the secret d and the ephemeral public key R from the ciphertext. Thus, $m = \mathsf{map}^{-1}(S - dR)$.

Q.2 One convenient way to map an element of $\mathbf{Z}_n$ to the elliptic curve is to multiple the integer by P. We define a function integer to convert a bitstring into an integer. I.e., $\mathsf{integer}(m) = \sum_{i=1}^{|m|} m_i 2^{|m|-i}$, where $|m|$ is the length of the bitstring m and m_i is the ith bit of m.
List the requirements on the map function to make the cryptosystem usable.
Say if the function $\mathsf{map}(m) = \mathsf{integer}(m)P$ satisfies them.

We need map *to*
- *map bitstrings to the group spanned by P (i.e., the entire elliptic curve, since P generates it by assumption),*
- *to be easy to compute,*
- *to be injective (to be invertible),*
- *and to have its inverse easy to compute.*

First of all, the output of map *is clearly in the group spanned by P.*
To make it invertible, we must restrict the length of m to $\log_2 n$.
Then, we realize that computing map^{-1} is hard since it consists of computing a discrete logarithm in the elliptic curve. So, this function is not usable.

Q.3 We now consider $\mathsf{map}(m) = (x, y)$ where $x = \mathsf{integer}(m)$, y is the smallest square root of $x^3 + ax + b$, and integer converts a bitstring into an integer. By reviewing the requirements on map, what do you think of this function?

If the computation works, map(m) *is clearly a point of the elliptic curve.*
To make it invertible, we must restrict the length of m *to* $\log_2 p$. *Then,* map$^{-1}(x,y) =$ integer$^{-1}(x)$. *So, we only have to convert an integer into a bitstring, which is easy to do.*
One problem is that $y^2 = x^3 + ax + b$ *has a solution if and only if* $x^3 + ax + b$ *is a quadratic residue, and this is not guaranteed. So,* m *may have no image by* map.

Q.4 Let k be a small (public) constant. We change the previous construction by taking x be the smallest integer at least equal to 2^kinteger(m) such that $x^3 + ax + b$ is a quadratic residue. Review again the required properties on map and provide algorithms to compute map and map^{-1}.

To evaluate map, *we increment* i *from* 0 *until* $\left(\frac{x^3+ax+b}{p}\right) = +1$, *for* $x = 2^k$integer$(m) + i$. *Then, we take* y *the smallest integer such that* $y^2 = x^3 + ax + b$ *and set* map$(m) = (x,y)$.
We define map$^{-1}(x,y) =$ integer$^{-1}\left(\lfloor \frac{x}{2^k} \rfloor\right)$.
This function seems to satisfy our needs, but the distribution of map(m) *is not so good (some points have no preimage), the complexity is not so nice (we may need to test quadratic residuosity for several values), and we may be careful that* map(map$^{-1}(x,y)$) *may not be equal to* (x,y).

Q.5 Assuming that p has 256 bits, propose a value (as small as possible) for k so that the previous construction should work with probability at least $1 - 2^{-80}$.
HINT: for this question, assume that $x \mapsto x^3 + ax + b$ maps intervals of size 2^k to "random values" in $\mathbf{Z}_p$.

A random value in $\mathbf{Z}_p$ *has a square root with probability roughly* $\frac{1}{2}$. *By rough estimates, a set of* 2^k *random values has all its elements with no square root with probability* 2^{-2^k}.
We can encrypt up to $p2^{-k}$ *possible plaintexts. Let* S *be the size of the plaintext space, between 2 and* $p2^{-k}$. *We have* S *sets of* 2^k *random values. Finally, the probability that one of the* S *set is like this is between* 2^{-2^k} *and* $S2^{-2^k}$, *depending on* S.
In the worst case, we have $S = p2^{-k}$. *We can just take* $k = 9$ *and have a probability lower than* 2^{-80} *in any case.*