Cryptography and Security — Final Exam
Solution

Serge Vaudenay

22.1.2025

- duration: 3h00
- 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
- answers should not be written with a pencil

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

1 Making Meaningful Collisions

We consider a hash function H following the Merkle-Damgård construction, to hash a file which is encoded like a programming language such as PDF or Postscript. We denote by $C(X, Y)$ the compression of a chaining value X with a message block Y to obtain the next chaining value.

Q.1 Describe an algorithm $\mathcal{A}(X) \rightarrow Y, Y'$ taking an initial chaining value X as input and producing two different message blocks Y and Y' such that $C(X, Y) = C(X, Y')$. Give its complexity in terms of number of compressions.
NOTE: full points only go to algorithms which are at least as good as the one in the solution.

We repeatedly try many different Y until we find such collision. For that, we use a dictionary and insert Y at index $C(X, Y)$. When we try a new Y', if $C(X, Y')$ is already in the dictionary, it defines Y such that $C(X, Y) = C(X, Y')$. Thanks to the birthday paradox, this algorithm eventually finds a collision with complexity $\mathcal{O}(2^{\frac{\ell}{2}})$ where ℓ is the hash length.
With MD5, we have $\ell = 128$ and the complexity corresponds to 2^{64} compressions.
With SHA256, we have $\ell = 256$ and the complexity corresponds to 2^{128} compressions.

Q.2 Give an algorithm using $\mathcal{A}$ as a subroutine and making a collision on H. The complexity of this algorithm must be equal to the one of $\mathcal{A}$ plus a constant overhead.

We let IV denote the initial chaining value of H. We run $\mathcal{A}(X) \rightarrow Y, Y'$. The single-block messages Y and Y' will make a collision on H. Indeed, Y and Y' has the same length so get the same padding Z. We have

$$H(Y) = C(C(\mathsf{IV}, Y), Z) = C(C(\mathsf{IV}, Y'), Z) = H(Y')$$

Q.3 By using a method seen in class, give an algorithm which takes two files F and F' as input and produce two files D and D' such that a document viewer would show D and F the same, it would show D' and F' the same, and we would have $H(D) = H(D')$.

We construct a block prefix *corresponding to the following program:*

```
1: read an address $x$ at address $\mathsf{addr}_1$
2: read a bit $b$ at address $x$
3: if $b = 0$ then jump at $\mathsf{addr}_2$
4: jump at $\mathsf{addr}_3$
```

where addr_1, addr_2, *and* addr_3 *point at places defined below. Given a single block* Y *and an address* x, *we construct a file* $f(Y, x) = \mathsf{prefix}\|Y\|x\|F\|F'$. *We define* addr_1 *as pointing to* addr_4. *We define* addr_2 *as pointing to* F. *We define* addr_3 *as pointing to* F'.
Then, we compute $X = C(\mathsf{IV}, \mathsf{prefix})$, *then run* $\mathcal{A}(X) \to Y, Y'$. *We define* addr_4 *be the address of a bit in which* $\mathsf{prefix}\|Y$ *and* $\mathsf{prefix}\|Y'$ *differ.*
When we run the viewer on $f(Y, \mathsf{addr}_4)$, *it will read the bit* b *at address* addr_4, *hence inside block* Y, *and jump to view either* F *(if* $b = 0$*) or* F' *(if* $b = 1$*). When we run the viewer on* $f(Y', \mathsf{addr}_4)$, *it will read* $\neg b$ *and view the other file.*
Finally, we define D *and* D' *be* $f(Y, \mathsf{addr}_4)$ *and* $f(Y', \mathsf{addr}_4)$ *in one order or the other so that* D *views* F. *We obtain that* D' *views* F'.
By construction, after compressing Y *or* Y', *there will be a collision. Since we concatenate with the same suffix, the collision will remain until the computation of* $H(D) = H(D')$. *Furthermore,*

Q.4 If now H follows the sponge construction, by defining a proper C function, show that the same method works.

A sponge state splits into $A\|B$ *where* A *has the same length as a block. We let* F *be the permutation over the state space in the sponge construction. By defining* $C(A\|B, Y) = F((A \oplus Y)\|B)$, *we express the sponge hash as an iterated compression following* C. *The same attack as before works.*

Q.5 What is the problem with the complexity of the previous method?

The problem is that now ℓ *is the state length, which is typically much larger than the hash length.*
With SHA3, we have $\ell = 1600$ *and the complexity is* 2^{800} *compressions.*

2 Secure KEM

We recall how that a KEM scheme is defined by three efficient algorithms

- $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Enc}(\mathsf{pk}) \to (k, \mathsf{ct})$
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to k'$

Q.1 Define correctness for a KEM.

We define the random variables pk*,* sk*,* k*,* ct*,* k' *by running*

```
1: $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$
2: $\mathsf{Enc}(\mathsf{pk}) \to (k, \mathsf{ct})$
3: $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to k'$
```

The KEM is (perfectly) correct if $\Pr[k = k'] = 1$.

Q.2 Adapt the IND-CPA notion for public-key cryptosystems to define the appropriate game for KEM.

The difference is that the plaintext cannot be chosen by the adversary. Instead, we have a kind of known-plaintext security. We provide to the adversary the plaintexts k_0 *and* k_1 *together with the ciphertext* ct_b.

Game Γ_b*:*

```
1: $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$
2: $\mathsf{Enc}(\mathsf{pk}) \to (k_0, \mathsf{ct}_0)$
3: $\mathsf{Enc}(\mathsf{pk}) \to (k_1, \mathsf{ct}_1)$
4: return $\mathcal{A}(\mathsf{pk}, k_0, k_1, \mathsf{ct}_b)$
```

Q.3 Define IND security using the previous game.

Informally, the KEM is IND-secure if for any efficient adversary $\mathcal{A}$*, the advantage* $\Pr[\Gamma_1 \to 1] - \Pr[\Gamma_0 \to 1]$ *is negligible.*
More precisely, the KEM is (t, ε)*-IND-secure if for any adversary* $\mathcal{A}$ *limited to a time complexity* t*, the advantage is bounded by* ε*.*

3 Quantum Billionaires

In the bitcoin infrastructure, a miner who succeeds to make a new block in the blockchain inserts a special transaction (called the coinbase transaction) at position zero. The input of the transaction consists of the collected fees from other transactions and some newly minted bitcoins. The output is typically the public key of the miner. There is no signature to validate the coinbase transaction.

For other transactions, there is normally an ECDSA signature. We use the standard elliptic curve P256 having a group generated by some point G of prime order n. Coordinates are modulo p. We have $p+1-2\sqrt{p} \leq n \leq p+1+2\sqrt{p}$. The transaction itself specifies some UTXO with the same public key Q and several outputs, where Q is in the group. If the transaction hashes onto a number z, a valid signature is a pair (r,s) such that $r = R_x \bmod n$, where R_x is the x-coordinate of $R = \frac{z}{s}G + \frac{r}{s}Q$.

Satoshi Nakamoto is believed to have created many address of miners which never spent the collected bitcoins in the early days of the blockchain. Overall, this sums up to one million bitcoins. (Note that the recent record rate was about 100 000USD per bitcoin.)

Q.1 The signature algorithm starts by computing $R = k \cdot G$ with a random k then it takes $r = R_x \bmod n$ which is part of the signature. Given r, how many possible points R on the curve would give $r = R_x \bmod n$.

Since $p+1-2\sqrt{p} \leq n \leq p+1+2\sqrt{p}$ and $R_x < p$, we can only have $R_x = r$ or $R_x = r+n$ if it is less than p. However, $r+2n \geq p$ for sure. So, we have up to two values for R_x. Given R_x, we have only two options for the value of R_y as it is the root of the equation which defines the elliptic curve. Hence, we have up to 4 possible R points.

Q.2 Recall how the signature algorithm works to sign z by using the signing key d and what is the relation between d and Q.

The keys d and Q are connected by the relation $Q = d \cdot G$. To sign, we pick a random k and compute $R = k \cdot G$. Then, r is set to the x-coordinate of R reduced modulo n. The value s is $s = \frac{z+dr}{k}$ modulo the order of the group.

Q.3 Imagine you have a quantum computer with enough qubits of memory. How would you collect the unspent bitcoins for yourself?

The first step is to collect the unspent coinbase outputs with a public key. They define points Q. Then, we can use the quantum computer to compute the discrete logarithm of every Q. Then, we can forge a signature to get those coins and pay our address.

Q.4 Instead of paying to a public key, we can pay to a public key hash without exposing the public key. How can a miner protect their revenue against quantum computer?

Instead of putting their Q in the coinbase output, they can put $H(Q)$. This way, the previous attack cannot collect Q. Inverting a hash function is not as easy as computing a discrete logarithm with a quantum computer.

Q.5 If a user signs any transaction, as soon as it appears on the blockchain, show that they can loose their remaining bitcoins in a similar attack even though they collected them by the pay-to-public-key-hash option.

Given a signature (r, s) we can compute the up to 4 possible R points. With each candidate for R, the adversary can compute a candidate for $Q = \frac{s}{r}R - \frac{z}{r}G$. The right Q will have the correct hash, so it can be easily found which candidate Q is the right Q. Once the right Q is isolated, the adversary can compute again the discrete logarithm and apply the previous attack.
The question was about stealing the bitcoins of a user which were not used to pay in the posted transaction (hence the UTXO of the user in previous transactions or the leftover paid to themselves in the current transaction). The point was to extract the point Q (supposed to be not public in the pay-to-public-hash technique) before breaking ECDSA.