Advanced Cryptography — Final Exam
Solution

Serge Vaudenay

21.6.2023

- duration: 3h
- any document allowed
- 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
- writing with pencil is not allowed

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

1 The Even-Mansour Cipher

This exercise is inspired from Dunkelmann-Keller-Shamir Minimalism in Cryptography: The Even-Mansour Scheme Revisited*, EUROCRYPT 2012, LNCS vol. 7237, Springer.*

In this exercise we consider a block cipher over n-bit blocks, which uses a $2n$-bit key (K_1, K_2) and defined by

$$\mathsf{Enc}_{K_1,K_2}(x) = \pi(x \oplus K_1) \oplus K_2$$

where π is a known permutation of the set $\{0,1\}^n$. In the adversarial model, the adversary is allowed to make D queries to a chosen plaintext/ciphertext oracle (that is, the adversary selects the direction for each query§ — either encryption or decryption — and thei input block, then gets either the encryption or the decryption of that block depending on the selected direction) and T queries to an oracle implementing π and π^{-1} (that is, the adversary selects the direction and the input and gets the image of that input by either π or π^{-1} depending on the selected direction). We consider key recover attacks: the goal of the adversary is to recover the hidden key (K_1, K_2).

Q.1 Let $\Delta \in \{0,1\}^n$ be a non-zero constant. We consider an adversary making D random pairs (x_i, x'_i), $i = 1, \ldots, D/2$, such that $x'_i \oplus x_i = \Delta$. The adversary makes D chosen plaintext queries to get $y_i = \mathsf{Enc}_{K_1,K_2}(x_i)$ and $y'_i = \mathsf{Enc}_{K_1,K_2}(x'_i)$, $i = 1, \ldots, D/2$. Then, the adversary takes T random pairs (u_j, u'_j), $j = 1, \ldots, T/2$, such that $u'_j \oplus u_j = \Delta$, and queries the other oracle to get $v_j = \pi(u_j)$ and $v'_j = \pi(u'_j)$, $j = 1, \ldots, T/2$.
How to select D and T to have good chances for a pair $(i.j)$ to exist such that $u_j = x_i \oplus K_1$?

This problem relates to the birthday paradox. The number of possible pairs is $\frac{D \times T}{4}$. For each pair, the probability that $u_j \oplus x_i$ is exactly Δ is 2^{-n}. Hence, we should take D and T such that $DT \approx 4 \cdot 2^n$. (Say DT proportional to 2^n.)

Q.2 How can the adversary isolate possible values for this pair (i, j) and estimate the expected number of incorrect values?

If $u_j = x_i \oplus K_1$, then $y_i = \pi(u_j) \oplus K_2 = v_j \oplus K_2$. Furthermore,

$$u'_j = u_j \oplus \Delta = x_i \oplus K_1 \oplus \Delta = x'_i \oplus K_1$$

Hence, $y'_i = \pi(u'_j) \oplus K_2 = v'_j \oplus K_2$. We deduce $y'_i \oplus y_i = v'_j \oplus v_j$.
The adversary can compute $y'_i \oplus y_i$ for every i, and $v'_j \oplus v_j$ for every j, then finds a match between the two lists to find a possible pair (i, j). For a pair which does not satisfy $u_j = x_i \oplus K_1$, the $y'_i \oplus y_i = v'_j \oplus v_j$ equality holds with probability 2^{-n}. So, we expect to find $\frac{DT}{4} 2^{-n}$ bad pairs in addition to the good one. With out selection $DT \approx 4 \cdot 2^n$, we expect to have one bad pair.

Q.3 Deduce a key recovery attack and estimate the success probability when DT is proportional to 2^n.

The adversary does the matching between the two lists as before and isolates possible values for (i,j). This suggests $K_1 = u_j \oplus x_i$ and $K_2 = v_j \oplus y_i$. This suggestion for (K_1, K_2) can be verified on any $(x_{i'}, y_{i'})$ pair to check whether the equation $y_{i'} = \pi(x_{i'} \oplus K_1) \oplus K_2$ is satisfied. This requires an additional query to π. Hence, the attack can rule out the bad pair and isolate the good one to deduce K_1 and K_2.

```
1: pick $\Delta, x_1, \ldots, x_{D/2}, u_1, \ldots, u_{T/2-1} \in \{0,1\}^n$ at random (with $\Delta \neq 0$)
2: set $x'_i = x_i \oplus \Delta$ and $u'_j = u_j \oplus \Delta$ for $i = 1, \ldots, D/2$ and $j = 1, \ldots, T/2-1$
3: query $y_i = \mathsf{Enc}(x_i)$ for $i = 1, \ldots, D/2$
4: query $y'_i = \mathsf{Enc}(x'_i)$ for $i = 1, \ldots, D/2$
5: query $v_i = \pi(u_j)$ for $j = 1, \ldots, T/2-1$
6: query $v'_v = \pi(u'_j)$ for $j = 1, \ldots, T/2-1$
7: makes a dictionary $H(y'_i \oplus y_i) = i$ for $i = 1, \ldots, D/2$
8: for $j = 1, \ldots, T/2-1$, checks if $H(v'_j \oplus v_j)$ exists and deduce a list $L$ of
   $(i,j)$ pairs
9: pick an arbitrary $i'$
10: for each $(i,j) \in L$ do
11:     set $K_1 = u_j \oplus x_i$ and $K_2 = v_j \oplus y_i$
12:     query $\pi(x_{i'} \oplus K_1)$
13:     if $y_{i'} \neq \pi(x_{i'} \oplus K_1) \oplus K_2$ then remove $(i,j)$ from $L$
14: end for
15: return $L$
```

With $DT = c2^n$, we expect to find $\frac{c}{4}$ good pairs (i,j) and $\frac{c}{4}$ bad pairs (i,j). We take $c = 4$ and the probability to have the good pair is

$$1 - \left(1 - 2^{-n}\right)^{\frac{DT}{4}} \approx 1 - e^{-\frac{DT}{4}2^{-n}} = 1 - e^{-1} \approx 63\%$$

2 Finding Heavy Differentials

This exercise is inspired from Dinur-Dunkelmann-Keller-Ronen-Shamir Efficient Detection of High Probability Statistical Properties of Cryptosystems via Surrogate Differentiation*, EUROCRYPT 2023, LNCS vol. 14007, Springer.*

Throughout this exercise, n denotes an integer and p denotes a probability. In asymptotic analysis, n goes to infinity and p may depend on n. Given a function $f : \{0,1\}^n \to \{0,1\}^n$ and $\alpha, \beta \in \{0,1\}^n$, we define $\mathsf{DP}_f(\alpha,\beta) = \Pr[f(X \oplus \alpha) = f(X) \oplus \beta$, where $\oplus$ is the bitwise exclusive OR and $X \in \{0,1\}^n$ is uniform. When x is such that $f(x \oplus \alpha) = f(x) \oplus \beta$, we say that x *follows* the characteristic (α,β). We say that (α,β) is a *heavy* characteristic if $\mathsf{DP}_f(\alpha,\beta) > p$. The objective of this exercise is to find heavy characteristics by having a black-box access to f and no other information about f. We assume that one memory register can store a value in $\{0,1\}^n$ and that an operation over elements of this set cost one unit of time complexity.

Q.1 Design an algorithm with oracle access to f which is able to find heavy characteristics with time complexity $\mathcal{O}(2^{2n})$ and memory $\mathcal{O}(2^{2n})$.

```
1: initialize an array $\mathsf{ct}[.,.]$ to 0
2: for $x$ and $\alpha$ in $\{0,1\}^n$ do
3:     $\beta \leftarrow f(x \oplus \alpha) \oplus f(x)$
4:     increment $\mathsf{ct}[\alpha,\beta]$
5: end for
6: initialize a list $L$ to empty
7: for $\alpha$ and $\beta$ in $\{0,1\}^n$ do
8:     if $\mathsf{ct}[\alpha,\beta] > p2^n$ then
9:         add $(\alpha,\beta)$ in $L$
10:    end if
11: end for
12: return $L$
```

Since this algorithm has two loops of 2^{2n} *iterations with elementary operations, this is the time complexity. Obviously, the first loop load* $\mathsf{ct}[.,.]$ *with* $\mathsf{ct}[\alpha,\beta] = 2^n.\mathsf{DP}_f(\alpha,\beta)$*. Hence, the second loop finds all heavy characteristics.*

Q.2 Given $\gamma \in \{0,1\}^n$, we define $g_\gamma(x) = f(x \oplus \gamma) \oplus f(x)$. We assume that when $X \in \{0,1\}^n$ is uniformly distributed, then the events "x follows (α,β)" and "$x \oplus \gamma$ follows (α,β)" are independent. When both events occur, we say that x is *good* for (α,β).
If (α,β) is heavy, prove that X is good for (α,β) with probability at least p^2 and that when such event occurs, then $g_\gamma(x) = g_\gamma(x \oplus \alpha)$.

We know that X follows the characteristic with probability at least p. We know that $X \oplus \alpha$ follows the characteristic with probability at least p as well, since $X \oplus \alpha$ follows the same distribution as X. Since we assume independence, X is good for the characteristic with probability p^2. When this happens,

$$g_\gamma(x \oplus \alpha) = f(x \oplus \alpha \oplus \gamma) \oplus f(x \oplus \alpha) = f(x \oplus \gamma) \oplus \beta \oplus f(x) \oplus \beta = g_\gamma(x)$$

Q.3 Given a heavy characteristic (α, β), if we pick $k = \left\lceil \sqrt{n} 2^{\frac{n}{2}} p^{-1} \right\rceil$ random values $x_1, \ldots, x_k$, show that except with negligible probability, there exist $\frac{n}{4}$ pairs (i, j) such thats $x_j = x_i \oplus \alpha$ and x_i is good. (Give a heuristic argument.)

Essentially, we have roughly $\frac{k^2}{2}$ pairs (i, j) with $i < j$. Each pair satisfies both conditions with probability $p^2 2^{-n}$. Hence, the expected number of pairs satisfying both conditions is $\frac{k^2}{2} p^2 2^{-n}$ which is $\frac{n}{2}$. It exceeds $\frac{n}{4}$ except with negligible probability.

Q.4 Complete the following algorithm and show that it can find heavy characteristics, except with negligible probability, and complexity lower than before. Precisely analyze the complexity.

```
1:  pick $x_1, \ldots, x_k \in \{0,1\}^n$ at random for $k = \left\lceil \sqrt{n} 2^{\frac{n}{2}} p^{-1} \right\rceil$
2:  initialize an array $\mathsf{Inv}[.]$ and the list $L$ to empty
3:  **for** $i = 1$ to $k$ **do**
4:      $y \leftarrow g_\gamma(x_i)$
5:      insert $x_i$ in the list $\mathsf{Inv}[y]$
6:      **if** $\mathsf{Inv}[y]$ has at least 2 elements **then** insert $y$ in $L$
7:  **end for**
8:  initialize $v\{.,.\}$ to an empty dictionary and $L'$ to the empty list
9:  **for** each $y$ in $L$ **do**
10:     **for** each $(x_i, x_j)$ pair of element of $\mathsf{Inv}[y]$ **do**
11:         $\alpha \leftarrow x_j \oplus x_i$, $\beta \leftarrow f(x_j) \oplus f(x_i)$
12:         **if** $v\{\alpha, \beta\}$ exists **then**
13:             $v\{\alpha, \beta\} \leftarrow v\{\alpha, \beta\} + 1$
14:         **else**
15:             $v\{\alpha, \beta\} \leftarrow 1$
16:         **end if**
17:         **if** $v\{\alpha, \beta\} \geq \frac{n}{4}$ **then** insert $(\alpha, \beta)$ in $L'$ and abort the **for** loop
18:     **end for**
19: **end for**
20: ...
```

Given that a heavy characteristic can be spotted by $\frac{n}{4}$ pairs, we can just look at the (α, β) pairs such that $v\{\alpha, \beta\} \geq \frac{n}{4}$ to get a list of candidates in L'. For each such candidate, we can approximate the probability of the characteristic using $\mathcal{O}(np^{-1})$ samples and isolate good candidates.

```
1: ...
2: for each $(\alpha, \beta)$ in $L'$ do
3:     initialize $a = 0$
4:     for $i = 1$ to $\lceil np^{-1} \rceil$ do
5:         pick a random $x \in \{0,1\}^n$
6:         increment $a$
7:         if $f(x \oplus \alpha) = f(x) \oplus \beta$ then increment $b$
8:     end for
9:     if $b/a > n/2$ then output $(\alpha, \beta)$
10: end for
```

*When the characteristic is heavy, we expect at least n values x to follow it so we find $n/2$ except with negligible probability. Likewise, characteristics of probability lower than $p/4$ will not be returned, except with negligible probability. The complexity of the first **for** loop is k. The second one iterate over up to k values of y. The inner **for** loop is bounded to $n/4$ iterations. Hence, the presented algorithm in the question has complexity $\mathcal{O}(nk)$. What we added has complexity $\mathcal{O}(np^{-1})$ times the size of L'. By showing that L' has size lower than $2^{\frac{n}{2}}$, the final complexity is $\mathcal{O}(n^{\frac{3}{2}} 2^{\frac{n}{2}} p^{-1})$.*

3 Blind Signatures

We consider a blind signature primitive which is defined by the following algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$ where λ is the security parameter;
- $\mathsf{SignC1}(\mathsf{pk}, m) \to (\mathsf{st}, \mathsf{query})$ where m is a message (bitstring);
- $\mathsf{SignS}(\mathsf{sk}, \mathsf{query}) \to \mathsf{resp}$;
- $\mathsf{SignC2}(\mathsf{st}, \mathsf{resp}) \to \sigma$;
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to \mathsf{true}/\mathsf{false}$.

When algorithms are executed in this order, correctness ensures that Verify returns true. The idea is that the signing process is run by the interaction between a client and a server. The server has the signing key sk and has authority to sign. The client knows which message m is to be signed but the server does not. The security notions are that signatures should be unforgeable (in a sense to specify in a question below) and query and σ should be unlinkable (in a sense to specify).

Q.1 Recall the EF-CMA security notions and explain why it does not fit to blind signatures.

In EF-CMA, the adversary can choose messages to be signed by an oracle. Here, the adversary would submit query *to a oracle implementing* $\mathsf{SignS}(\mathsf{sk}, .)$. *But* query *can be made following whatever strategy by the adversary and is supposed to be unlinkable to the signed message. Hence, whenever terminates with a forgery candidate (m, σ), there is no way to know if indeed the adversary followed the protocol to make the oracle blindly sign m.*

Q.2 We try to formalize unforgeability by the notion of one-more forgeries. Following this game, the adversary wins by showing more signed messages than the number of queries to a $\mathsf{SignS}(\mathsf{sk}, .)$ oracle. Properly define the one-more forgery game and formalize security with respect to this notion.

We say that the blind signature is one-more unforgeable if for any PPT $\mathcal{A}$, the probability that the following game outputs 1 is a negligible function of λ.

```
Input: $\lambda$
 1: $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$
 2: $q \leftarrow 0$
 3: $\mathcal{A}^{\mathcal{O}}(\mathsf{pk}) \to (m_i, \sigma_i)_{i=1,\ldots}$
 4: if the number of $i$ is not larger than $q$ then abort
 5: if there exists $i$ and $j$ such that $i < j$ and $m_i = m_j$ then abort
 6: if there exists $i$ such that $\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i) = \mathsf{false}$ then abort
 7: return 1

Oracle $\mathcal{O}(\mathsf{query})$:
 8: increment $q$
 9: $\mathsf{SignS}(\mathsf{sk}, \mathsf{query}) \to \mathsf{resp}$
10: return resp
```

Q.3 Formalize the notion of unlinkability, where the adversary is now the server.

We limit to honest key generation here. (Allowing the adversary to create pk *maliciously would be more complicated.)*

```
Input: $\lambda$ and a bit $b$
 1: $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$
 2: $\mathcal{A}_1(\mathsf{pk}) \to z$
 3: return $z$

Oracle $\mathsf{C1}(i, m)$:
 4: if $\mathsf{query}_i$ defined then abort
 5: $\mathsf{SignC1}(\mathsf{pk}, m) \to (s_i, \mathsf{query}_i)$
 6: return $\mathsf{query}_i$

Oracle $\mathsf{C2}(i, \mathsf{resp})$:
 7: if $s_i$ undefined or $\sigma_i$ defined then abort
 8: $\mathsf{SignC2}(s_i, \mathsf{resp}) \to \sigma_i$
 9: return $\sigma_i$

Oracle $\mathsf{Chal1}(m_0, m_1)$:
10: if $\mathsf{query}'_0$ defined then abort
11: $\mathsf{SignC1}(\mathsf{pk}, m_b) \to (s'_0, \mathsf{query}'_0)$
12: $\mathsf{SignC1}(\mathsf{pk}, m_{1-b}) \to (s'_1, \mathsf{query}'_1)$
13: return $(\mathsf{query}'_0, \mathsf{query}'_1)$

Oracle $\mathsf{Chal2}(\mathsf{resp}_0, \mathsf{resp}_1)$:
14: if $s'_0$ undefined or $\sigma'_0$ defined then abort
15: $\mathsf{SignC2}(s'_0, \mathsf{resp}_b) \to \sigma'_0$
16: $\mathsf{SignC2}(s'_1, \mathsf{resp}_{1-b}) \to \sigma'_1$
17: return $(\sigma'_0, \sigma'_1)$
```

The advantage is the difference of the probability that the game returns 1 when $b = 0$ and $b = 1$. We say that the blind signature is unlinkable if for every PPT $\mathcal{A}$, the advantage is a negligible function of λ.

In the game, we modelled the client with oracles. The adversary specifies the session index i of the signature session and can choose the message to be signed. Some challenge oracles use two special sessions which are either permuted or not. The adversary must figure out is the challenge sessions are permuted.

Q.4 We tweak RSA so that it fits the notion of blind signature. We define KeyGen as in RSA and $\mathsf{SignS}(\mathsf{sk}, \mathsf{query}) = \mathsf{query}^d \bmod N$, where $\mathsf{sk} = (N, d)$. Propose some algorithms for SignC1, SignC2, and Verify in order to obtain a blind signature which is one-time unforgeable and unlinkable. (Give arguments for the security, no formal proof is required but insecure solutions will have a lower grade.)

RSA signatures can be transformed into other RSA signatures, because of the homomorphic property. To avoid this, we use the approach of the full-domain hash: $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *checks that* $\sigma^e \bmod N = H(m)$, *where* $\mathsf{pk} = (N, e)$ *and* H *is a random oracle giving outputs in* $\mathbf{Z}_N$. *This way, valid signatures for a set of message cannot be transformed into a valid signature for a new message. To make the signature unlinkable, we blind the message to be signed by using a multiplicative mask* r*:*

```
SignC1(pk, m):
  1: pick $r \in \mathbf{Z}_N^*$ at random
  2: $\mathsf{query} \leftarrow (H(m)r^e) \bmod N$
  3: $\mathsf{st} \leftarrow (\mathsf{pk}, r)$
  4: return $(\mathsf{st}, \mathsf{query})$

SignC2(st, resp):
  5: parse $\mathsf{st} \rightarrow ((N, e), r)$
  6: $\sigma \leftarrow (\mathsf{resp}/r) \bmod N$
  7: return $\sigma$
```