CS-E4340 Cryptography: Exercise Sheet 9

Estuardo Alpirez Bock, Christopher Brzuska and Kirthivaasan Puniamurthy

Aalto University, Finland

Abstract. This exercise deals with definitional studies for white-box cryptography and leakage resilient cryptography. In Definition 1, we provide a general definition for a white-box compiler. This compiler takes as input the secret key of a symmetric encryption scheme, and outputs a white-box encryption program. The white-box encryption program takes as input a message m and a nonce nc, and outputs an encryption of m. The white-box program is functionally equivalent to $\mathsf{enc}(k, m, nc)$.

Exercise 1 (Side-channel analysis)**.** Below we see a description of the Montgomery Ladder using projective coordinates. This algorithm is a popular choice for implementing the scalar multiplication in elliptic curve cryptosystems implemented in hardware. We recall that in such cryptosystems, the scalar k corresponds to the secret key and the multiplication kP is performed, e.g., for generating a signature or performing encryptions. The algorithm goes through an initialization phase (line 1), where 4 registers are set with some initial values. The algorithm then enters a loop (lines 2 to 10). The loop is executed once for each bit of the scalar k and depending on the bit value we execute either lines 4 and 5 or lines 7 and 8. Note that in both cases, the loop consists of the same arithmetic operations (multiplication, addition, squaring and overwriting the value of a register).

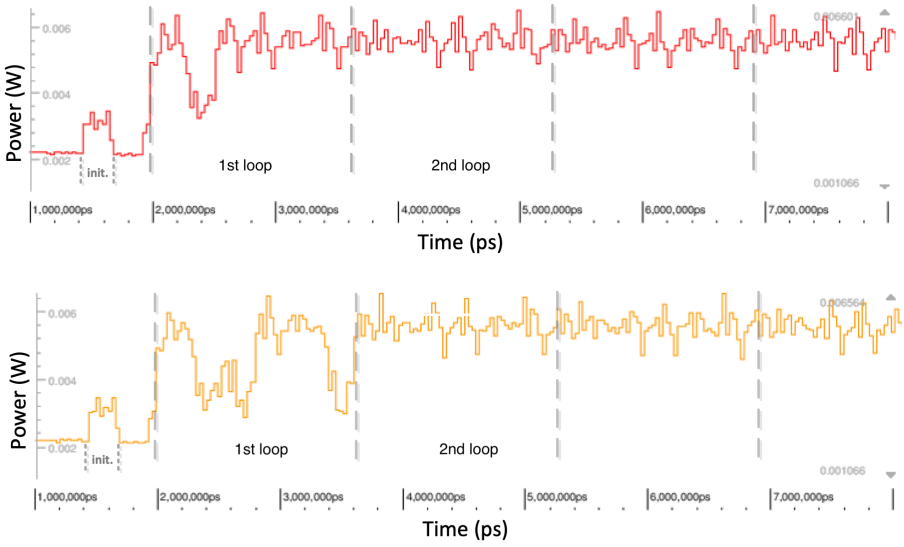
Figure 1 shows two power traces obtained during the execution of the algorithm (these power traces correspond actually to the initialization period and to the first 4 loop iterations of the algorithm). For the power trace above, a scalar $k = \kappa_1$ was used, while for the power trace below, a different scalar $k = \kappa_2$ was used. From observing these power traces, are you able to extract any bits from any of the two scalars?

Algorithm 1 Montgomery algorithm for the kP-operation using projective coordinates

Input: $k = (k_{l-1}, ..., k_1, k_0)_2$ with $k_{l-1} = 1, P = (x, y) \in E(GF(2^m))$.
Output: $kP = (x_1, y_1)$.
1: $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$.
2: **for** i from $l-2$ downto 0 **do**
3: **if** $k_i = 1$ **then**
4: $T \leftarrow Z_1$, $Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$, $X_1 \leftarrow xZ_1 + X_1 X_2 T Z_2$,
5: $T \leftarrow X_2$, $X_2 \leftarrow X_2^4 + bZ_2^4$, $Z_2 \leftarrow T^2 Z_2^2$.
6: **else**
7: $T \leftarrow Z_2$, $Z_2 \leftarrow (X_2 Z_1 + X_1 Z_2)^2$, $X_2 \leftarrow xZ_2 + X_1 X_2 T Z_1$,
8: $T \leftarrow X_1$, $X_1 \leftarrow X_1^4 + bZ_1^4$, $Z_1 \leftarrow T^2 Z_1^2$.
9: **end if**
10: **end for**
11: $x_1 \leftarrow X_1/Z_1$.
12: $y_1 \leftarrow y + (x + x_1)[X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1 Z_2)]/(xZ_1 Z_2)$.
13: **return** $((x_1, y_1))$.

Hints:

1. l is the length of the key, e.g. 233 bits. In this context, we say that the first bit of the key (k_{l-1}) is always a 1. The loop iterations start when we read the second bit (k_{l-2}).



2. $P = (x, y)$ is a point on the elliptic curve, the value of its coordinate x is used in line 1 to initialize some registers (e.g. register X_1). b is a coefficient used for defining the elliptic curve.
3. During the first loop iteration, the registers (or variables) X_1, X_2, Z_1 and Z_2 will be initialized with the values set on line 1.
4. Note that the Montgomery Ladder is expected to provide robustness against simple power analysis attacks. Thus, you are not really expected to extract *many* bits just from observing these two power traces. Note that most loop iterations look very similar and it is difficult to determine any key dependencies. However the first loop iteration looks different than the rest. Moreover, the first loop iteration for both power traces looks very different.
5. Recall that the power consumption of a device depends on the number of gates being switched during the computation. Thus, power consumption (per clock cycle) is highly dependent on the type of operations we are performing and also on the inputs we are processing. Do you think all multiplications demand equally high power? Or does it depend on the values we are multiplying?

Solution 1. For both power traces, we are only able to extract the key bit processed during the first loop. For the upper power trace, a key bit with value 1 was processed. For the lower trace, a key bit with value 0 was processed.
The reason we are able to distinguish is because of how the registers are initialized on line 1 of the algorithm. As we can see, the register Z_1 is initialized with a value 1. In both possible loop iterations, Z_1 is used for performing one or more multiplications. However any multiplication performed with register $Z_1 = 1$ will not be an expensive operation (i.e. it is a multiplication times 1), contrary to a multiplication performed with two values bigger than 1.
In the loop iteration for a key bit with value 1, the register Z_1 is multiplied once with the register X_2, then Z_1 is overwritten. That is, we only have one multiplication with $Z_1 = 1$ for this loop iteration, hence the one dip in the upper power trace.
In the loop iteration for a key bit with value 0, the register Z_1 is multiplied once with the register X_2, then it is multiplied with $X_1 X_2 T$, it is squared twice, multiplied with b and only in the end the register is overwritten. Thus we have more than one multiplication

performed with a value equal to 1, and we will have more than one operation which consumes a notably low amount of power. Hence we obtain more of such dips

Definition (White-box compiler)**.** An algorithm Comp is called a white-box compiler for a symmetric encryption scheme $\mathsf{se} = (\mathsf{se.enc}, \mathsf{se.dec})$ if for all key values $k \in \{0,1\}^\lambda$, all nonces $nc \in \{0,1\}^\lambda$ and all messages $m \in \{0,1\}^*$, we have $\Pr[\mathsf{se.enc}(k, m, nc) = \mathsf{WB}(m, nc)] = 1$, where the probability is taken over $\mathsf{WB} \leftarrow\!\!\$\ \mathsf{Comp}(k)$.

Exercise 2 (Security against key extraction)**.** In the lecture we discussed how white-box programs may be susceptible to key-extraction attacks. Achieving security against key extraction is crucial for a white-box program. However as we will see in this exercise, white-box programs need to achieve further properties in order to provide any meaningful security.

Below we provide a definition capturing the property of security against key extraction for white-box programs.[1] Explain that while this definition captures security against key extraction, it does not really capture useful properties we would want from a white-box (cryptographic) program. Give an example of a program which achieves security against key extraction (i.e. is secure in the model below), yet it does not provide any further meaningful security (e.g. confidentiality of plaintexts). For this, you may design a secure symmetric-encryption scheme $\mathsf{se}_{\text{example}}$ (assuming an existing secure symmetric-encryption scheme se) and a compiler comp for $\mathsf{se}_{\text{example}}$ such that the compiler for $\mathsf{se}_{\text{example}}$ outputs a white-box encryption program which is (1) functional equivalent to the encryption of $\mathsf{se}_{\text{example}}$, (2) is secure against key extraction and (3) does not achieve any meaningful security in the presence of a white-box adversary.

Hint: Note that the definition below follows a notation style similar to the security definition for one-way functions (see Section 3.1 on the DAF (Security definition 1)).

Definition (Security against key extraction)**.** Let $\mathsf{Exp}^{\mathsf{Kextr}}_{\mathsf{WB},\mathcal{A}}(1^\lambda)$ be a security experiment defined by

$\underline{\mathsf{Exp}^{\mathsf{Kextr}}_{\mathsf{Comp},\mathcal{A}}(1^\lambda)}$
$k \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\mathsf{WB} \leftarrow\!\!\$\ \mathsf{Comp}(k)$
$k^* \leftarrow\!\!\$\ \mathcal{A}(\mathsf{WB})$
if $k^* = k$ **then**
  return 1
return 0

A white-box compiler Comp for a symmetric encryption scheme se is secure against key extraction if for all PPT adversaries $\mathcal{A}$ the winning probability $\mathsf{Win}^{\mathsf{Kextr}}_{\mathsf{Comp},\mathcal{A}}(1^\lambda) := \Pr\left[1 = \mathsf{Exp}^{\mathsf{Kextr}}_{\mathsf{Comp},\mathcal{A}}(1^\lambda)\right]$ is negligible in λ. (See discussion of search games in the DAF.)

Hints:

1. It may be useful to think of other primitives (besides encryption programs) which we may want to white-box. For instance, suppose you want to white-box a MAC program. As we recall from previous lectures, a good MAC should achieve an unforgeability property. Suppose we white-box a *bad* MAC program which does not achieve the unforgeability property. Suppose however, that the white-box version of this program manages to hide its key such

[1] In the literature, this definition might be referred to as *Unbreakability*. The name Unbreakability derives from the fact that when we manage to extract the key of a cryptographic implementation, we say that we *break* the implementation.

that it is indeed very difficult for an adversary to recover it from the implementation. Would such a MAC program be considered secure in the definition above, even though we know that it is a bad MAC?

Solution 2. We want to show that even if a white-box program hides its symmetric encryption key, an adversary might still be able to break further security properties we would usually desire. We simply define a symmetric encryption key that uses only half of its secret key. Then, we can throw away the other half (so that the key cannot be recovered since half of it is lost), but the scheme is still terribly insecure. It is impossible for the adversary to recover the complete key since only half of the key is located within the implementation and the other half is not really used.
Another example would be a scheme which only encrypts one half of the message and outputs the other half of the message in plain. Such a scheme would not achieve any notion of confientiality. But if the adversary is unable to recover the secret key from the white-box program, the scheme would be considered secure in the definition above.

Exercise 3 (Leakage resilient cryptography)**.** In wake of the evident threat of side-channel attacks, a research area known as *leakage resilient cryptography* was introduced. This area covers definitional studies considering an adversary who obtains some leakage of the secret key used within a cryptographic scheme. Then, the adversary tries to break security of the cryptographic scheme, given the leakage.

a) Provide a definition for IND-CPA security of an encryption scheme under leakage. This definition should look similar to the traditional IND-CPA definition we have seen in previous lectures, with some additional steps where the adversary obtains leakage from the secret key.
b) What do you think may be practical ways of achieving security under leakage?

Hints:

1. In definitions for cryptographic schemes secure under leakage, the adversary usually defines his own leakage function. However for the definition to make sense, the leakage function should have some restrictions. E.g. the leakage function should not be able to provide the adversary with the complete value of the secret key, else the adversary would always trivially win the security games. I.e. there may be some output length restrictions for the leakage functions. Alternatively, there might not be length restrictions for the output of the leakage functions, but the adversary should not be able to trivially derive the value of the secret key from the leakage. How would you capture these properties of the leakage function within the definition? What would be a reasonable output length restriction?

Solution 3. There are different ways of capturing IND-CPA security under leakage. Usually we have an adversary defining a leakage function in the beginning of the game, then the leakage function is applied to the secret key and its output is given to the adversary. The leakage function can be randomized, e.g. it can be a function which randomly selects some bits of the secret key and just returns them to the adversary. Then, given the leakage, the adversary proceeds to play a normal IND-CPA game. At some part of the game, we need to check whether the leakage is valid. I.e. whether the leakage is not too long or whether the secret key still has enough min-entropy, even given the leakage.
Below we have an example of such a security definition. The adversary defines the leakage function and then the leakage function is applied to the key k. Then, the leakage is given

to the adversary and the adversary plays a classic IND-CPA game. In the third to last line, we check if the secret key still has enough min-entropy, given the leakage.

$\underline{\mathsf{IND}-\mathsf{CPA}_{\mu,\mathcal{A}}(1^n)}$

$b \leftarrow\$ \{0,1\}$
$k \leftarrow\$ \mathsf{kgen}(1^n)$
$\mathsf{leak}(\cdot) \leftarrow\$ \mathcal{A}(1^n)$
$lk \leftarrow\$ \mathsf{leak}(k)$
$b' \leftarrow\$ \mathcal{A}^O(lk)$
if $b' = b$ **and** $(\mathsf{leak}(k)|H_\infty(k)) \geq \mu)$
  return 1
else return 0

$\underline{O(m_0, m_1)}$

$c_b \leftarrow\$ \mathtt{Enc}(k, m_b)$
return c_b

A trivial, practical way of increasing the robustness against leakage adversaries is by using keys of very large size. If the key is too large, the adversary might not be able to extract enough bits in order to break security. However the use of large keys within cryptographic schemes should not affect the efficiency of the schemes. One common strategy is to have a very large key but use only some randomly chosen bits from it for each encryption.

Exercise 4 (Impossibility)**.** AES has been shown to be tremendously difficult to white-box. As we will see via an example in this exercise, some symmetric encryption schemes are actually impossible to be securely white-boxed, and they might not even achieve the modest goal of security against key extraction. Let se be an IND-CPA secure symmetric encryption scheme, let f be a pseudorandom function and let se' be the symmetric encryption scheme as defined below. Let comp be a (correct) white-box compiler for se', i.e. $\mathsf{WB} \leftarrow\$ \mathsf{Comp}(k'_{\mathsf{se}})$.

Give the code of an efficient attacker $\mathcal{A}$ such that $\mathsf{Win}^{\mathsf{Kextr}}_{\mathsf{Comp},\mathcal{A}}(1^\lambda) := \Pr\left[1 = \mathsf{Exp}^{\mathsf{Kextr}}_{\mathsf{Comp},\mathcal{A}}(1^\lambda)\right]$ is non-negligible. You only need to argue about the success probability informally.

Hint: A black-box program means that an adversary is able to obtain input/output pairs but not know the code/internals of the program. What happens when an adversary instead interacts with a white-boxed program, i.e. it knows the program's code/internals, although it cannot make sense of it?

$\underline{\mathsf{kgen}'(1^\lambda)}$

$k_{\mathsf{SE}} \leftarrow\$ \{0,1\}^\lambda$
$k_x \leftarrow\$ \{0,1\}^\lambda$
$k_{\mathsf{nc}} \leftarrow\$ \{0,1\}^\lambda$
$k_{\mathsf{se}'} \leftarrow k_{\mathsf{SE}}||k_x||k_{\mathsf{nc}}$
return $k_{\mathsf{se}'}$

$\underline{\mathsf{enc}'(k_{\mathsf{se}'}, x, nc)}$

$\mathsf{C} \leftarrow \mathtt{PARSE}(x)$
$k_{\mathsf{SE}}||k_x||k_{\mathsf{nc}} \leftarrow k_{\mathsf{se}'}$
$d \leftarrow 1$
for i **from** 1 **to** λ
  $x_i \leftarrow \mathtt{PRF}(k_x, \mathsf{bin}_\lambda(i)||\mathsf{nc})$
  $\mathsf{nc}_i \leftarrow \mathtt{PRF}(k_{\mathsf{nc}}, \mathsf{bin}_\lambda(i)||\mathsf{nc})$
  if $\mathsf{C}(x_i, \mathsf{nc}_i) = 0||\mathsf{enc}(k_{\mathsf{SE}}, x_i, \mathsf{nc}_i)$
    $d \leftarrow d \wedge 1$
  else $d \leftarrow d \wedge 0$
$c' \leftarrow \mathsf{enc}(k_{\mathsf{SE}}, x, nc)$
if $d = 0$
  $c \leftarrow 0||c'$
else $c \leftarrow 1||c'||k_{\mathsf{se}'}$
return c

$\underline{\mathsf{dec}'(k_{\mathsf{se}'}, c, \mathsf{nc})}$

$d||\widetilde{c} \leftarrow c$
$k_{\mathsf{SE}}||k_x||k_{\mathsf{nc}} \leftarrow k_{\mathsf{se}'}$
if $d = 0$
  $x \leftarrow \mathsf{dec}(k_{\mathsf{SE}}, \widetilde{c}, \mathsf{nc})$
else
  $c'||k' \leftarrow \widetilde{c}$
  if $k' = k_{\mathsf{se}'}$
    $x \leftarrow \mathsf{dec}(k_{\mathsf{SE}}, c', \mathsf{nc})$
  else $x \leftarrow \perp$
return x

Solution 4. The key observation is that the encryption scheme outputs its secret key k, if the message given as input corresponds to a circuit describing the encryption algorithm (including its secret key). In the black-box attack model, an adversary cannot provide such an input since he does not have access to such a circuit, the adversary only sees the input/output behaviour of the program. Thus, this scheme is secure in the black-box attack model. In the white-box attack model however, the adversary gets access to the code decribing the encryption program (even if he does not understand what's written there). Thus the adversary can copy that code, parse it into a message and provide it as input to the encryption program. The encryption program will leak its symmetric key. The adversary proceeds as follows.

$\mathcal{A}(\mathsf{WB})$

$x \leftarrow \texttt{PARSE}(\mathsf{WB})$ // use the code describing WB as the message

$\mathsf{nc} \leftarrow\!\!\$\ \{0,1\}^\lambda$

$c \leftarrow \mathsf{WB}(x, \mathsf{nc})$

$1||c'||k_{\mathsf{se}'} \leftarrow c$

$k_{\mathsf{SE}}||k_x||k_{\mathsf{nc}} \leftarrow k_{\mathsf{se}'}$

return k_{SE}

It follows from the correctness of se' that the adversary wins the key extraction game with overwhelming probability.

Exercise 5 (White-box as PKE)**.** One possible view on security of white-box cryptography (without hardware-binding) is to build public key encryption scheme from a white-box of symmetric encryption scheme and demand that the resulting PKE satisfies IND-CPA security of a PKE. Given a white-box symmetric encryption scheme, describe a pke based on it and argue intuitively why it satisfies IND-CPA of PKE.

Solution 5. Key generation samples a random key, runs the compiler, outputs the key k as secret key and the white-box program as public-key. Encryption uses the white-box program. Decryption just uses k. Anybody can encrypt using the publicly known white-box program, but they cannot decrypt since they cannot extract the key k from it. Using IND-CPA security as a security definition of the compiler now demands that the white-box encryption program does not harm confidentiality of ciphertexts and, in particular, that the attacker cannot use the white-box encryption program for decrypting. This clearly implies that an adversary should not be able to extract the secret key from the white-box program (else, he could use that key for decrypting ciphertexts and breaking confidentiality).

Exercise 6 (Hardware-binding)**.** In the lecture we discussed the issue that white-box programs might be susceptible to re-distribution attacks (or *code-lifting* attacks). In such attacks, an adversary copies the complete white-box program and simply uses it on a device of its choice. One method to mitigate such attacks is to compile the white-box program such that it can only be executed on one specific device. We refer to this property as hardware-binding. One way to implement hardware-binding is by having a program which tries to check on which device it is running before performing a computation. For instance, if the device running the white-box has a secure hardware module (which is not accessible to the white-box adversary), we could use one functionality of that module to check if we are running on the desired hardware.

Consider now a key derivation function (KDF) that we wish to white-box. Here, we consider a simple key derivation function which uses a secret key $k_{\mathtt{kdf}}$ to derive further keys from a context value e. The derived keys should be indistinguishable from random and thus, the syntax and security definition of this KDF are equal to those we have defined for PRFs[2]. In this exercise we will study a security definition for a white-box KDF with hardware-binding. Below, we first introduce the syntax for our hardware module. Then we introduce the syntax for the white-box KDF. Finally, we provide a definition for the security of a white-box KDF.

We observe that the white-box KDF works in collaboration from a response coming from the hardware module. Without this response, the white-box KDF cannot compute anything. The security definition has some subtle flaws. Namely, currently, an adversary can win this game. Can you spot the flaws, explain them and correct them?

Definition (Hardware Module HWM)**.** A *hardware module* HWM consists of four algorithms $(\mathtt{kgen}_{\mathtt{HW}}, \mathtt{SubKgen}_{\mathtt{HW}}, \mathtt{Resp}_{\mathtt{HW}}, \mathtt{Check}_{\mathtt{SW}})$, where $\mathtt{kgen}_{\mathtt{HW}}$ is a PPT algorithm, and the algorithms $\mathtt{SubKgen}_{\mathtt{HW}}$, $\mathtt{Resp}_{\mathtt{HW}}$ and $\mathtt{Check}_{\mathtt{SW}}$ are deterministic algorithms with the following syntax:

$$k_{\mathtt{HWm}} \leftarrow\!\$\ \mathtt{kgen}_{\mathtt{HW}}(1^n) \qquad k_{\mathtt{HWs}} \leftarrow \mathtt{SubKgen}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label})$$
$$\sigma \leftarrow \mathtt{Resp}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label}, x) \qquad b \leftarrow \mathtt{Check}_{\mathtt{SW}}(k_{\mathtt{HWs}}, x, \sigma),$$

Correctness requires that for all security parameters $n \in \mathbb{N}$,

$$\Pr[\mathtt{Check}_{\mathtt{SW}}(\mathtt{SubKgen}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label}), x, \mathtt{Resp}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label}, x)) = 1] = 1,$$

where the probability is over the sampling of $k_{\mathtt{HWm}} \leftarrow\!\$\ \mathtt{kgen}_{\mathtt{HW}}(1^n)$.

The purposes of the algorithms are:

- The algorithm $\mathtt{kgen}_{\mathtt{HW}}$ is used within the HWM to generate a main key $k_{\mathtt{HWm}}$, which is located in the HWM.
- The algorithm $\mathtt{SubKgen}_{\mathtt{HW}}$ is used within the HWM to generate a subkey $k_{\mathtt{HWs}}$, which is used when compiling the whitebox program (the whitebox program can later use the key $k_{\mathtt{HWs}}$ to verify that it is running in the correct hardware.)
- The algorithm $\mathtt{Resp}_{\mathtt{HW}}$ is run in the HWM. The whitebox program can ask the HWM to give a response on which hardware it is running, that invokes $\mathtt{Resp}_{\mathtt{HW}}$ algorithm which gives the answer.
- The algorithm $\mathtt{Check}_{\mathtt{SW}}$ is used within the whitebox program to check that the certificate given by $\mathtt{Resp}_{\mathtt{HW}}$ was valid. To do this, $\mathtt{Check}_{\mathtt{SW}}$ uses the subkey $k_{\mathtt{HWs}}$.

Definition (WBHW)**.** A *white-box key derivation scheme with hardware binding* WBHW consists of a *hardware module* HWM, a *key derivation function* KDF, and a PPT compiling algorithm Comp:

$$\mathtt{WKDF} \leftarrow\!\$\ \mathsf{Comp}(k_{\mathtt{kdf}}, k_{\mathtt{HWs}}).$$

For all genuine $k_{\mathtt{HWm}}$, for all $k_{\mathtt{kdf}}$, for all *label*, for all e, for all $k_{\mathtt{HWs}} = \mathtt{SubKgen}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label})$ and $\sigma = \mathtt{Resp}_{\mathtt{HW}}(k_{\mathtt{HWm}}, \mathit{label}, e)$, we have

$$\Pr[\mathsf{KDF}(k_{\mathtt{kdf}}, e) = \mathtt{WKDF}(e, \sigma)] = 1$$

where the probability is taken over compiling $\mathtt{WKDF} \leftarrow\!\$\ \mathsf{Comp}(k_{\mathtt{kdf}}, k_{\mathtt{HWs}})$.

Definition. A hardware-bound white-box key derivation scheme WBHW is IND-WKDF-secure if the real and ideal games $\mathtt{Gind\text{-}wkdf}^b_{\mathsf{WBHW}}$ are computationallly indistinguishable, that is, for all PPT adversaries $\mathcal{A}$ the advantage

$$\mathbf{Adv}^{\mathtt{Gind\text{-}wkdf}^0_{\mathsf{WBHW}},\mathtt{Gind\text{-}wkdf}^1_{\mathsf{WBHW}}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[1 = \mathcal{A} \circ \mathtt{Gind\text{-}wkdf}^0_{\mathsf{WBHW}}\right] - \Pr\left[1 = \mathcal{A} \circ \mathtt{Gind\text{-}wkdf}^1_{\mathsf{WBHW}}\right]\right|$$

is negligible.

[2] We here use the term KDF, because it makes more sense conceptually, but if it is confusing, think PRF each time you read KDF.

$\texttt{Gind-wkdf}^0_{\mathsf{WBHW}}$

Parameters

λ:	security parameter
WBHW:	WKDF scheme

Package State

$k_{\texttt{kdf}}$:	KDF key
$k_{\texttt{HWm}}$:	hardware main key
$k_{\texttt{HWs}}$:	hardware sub-key

GETWB($label$)

assert $\texttt{WKDF} \neq \bot$
$k_{\texttt{kdf}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$k_{\texttt{HWm}} \leftarrow\!\!\$\ \mathsf{kgen}_{\texttt{HW}}(1^\lambda)$
$k_{\texttt{HWs}} \leftarrow \mathtt{SubKgen}_{\texttt{HW}}(k_{\texttt{HWm}}, label)$
$\texttt{WKDF} \leftarrow\!\!\$\ \mathsf{Comp}(k_{\texttt{kdf}}, k_{\texttt{HWs}})$
return WKDF

KDF()

$e \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\hat{k} \leftarrow \mathsf{KDF}(k_{\texttt{kdf}}, e)$
return $(\hat{k}, e)$

Resp(e, $label$)

$\sigma \leftarrow \mathtt{Resp}_{\texttt{HW}}(k_{\texttt{HWm}}, label, e)$
return σ

$\texttt{Gind-wkdf}^1_{\mathsf{WBHW}}$

Parameters

λ:	security parameter
WBHW:	WKDF scheme

Package State

$k_{\texttt{kdf}}$:	KDF key
$k_{\texttt{HWm}}$:	hardware main key
$k_{\texttt{HWs}}$:	hardware sub-key

GETWB($label$)

assert $\texttt{WKDF} \neq \bot$
$k_{\texttt{kdf}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$k_{\texttt{HWm}} \leftarrow\!\!\$\ \mathsf{kgen}_{\texttt{HW}}(1^\lambda)$
$k_{\texttt{HWs}} \leftarrow \mathtt{SubKgen}_{\texttt{HW}}(k_{\texttt{HWm}}, label)$
$\texttt{WKDF} \leftarrow\!\!\$\ \mathsf{Comp}(k_{\texttt{kdf}}, k_{\texttt{HWs}})$
return WKDF

KDF()

$e \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\hat{k} \leftarrow\!\!\$\ \{0,1\}^\lambda$
return $(\hat{k}, e)$

Resp(e, $label$)

$\sigma \leftarrow \mathtt{Resp}_{\texttt{HW}}(k_{\texttt{HWm}}, label, e)$
return σ

Solution 6. The whiteboxed KDF takes as input e and σ where σ is the response from the hardware module that tells whether the hardware is correct.
Now, if the adversary wants to win the game, the adversary should be able to produce a fake σ that convinces the whiteboxed KDF even though σ was generated by the adversary and not by the hardware module.
The current security definition allows the adversary to simply query the correct σ from Resp oracle.

$\mathcal{A}$

$\text{WKDF} \leftarrow \mathsf{GETWB}(l)$
$(k, e) \leftarrow \mathsf{KDF}$
$\sigma \leftarrow \mathsf{Resp}(e, l)$
if $k = \text{WKDF}(e, \sigma)$
  return 1
else
  return 0

This adversary always returns 1 in the real game, while in the ideal game the key k is random, and hence adversary returns 1 with probability $2^{-\lambda}$, so the adversary has advantage $1 - 2^{-\lambda}$ which is non-negligible (almost one!).
We can easily modify the game by integrating a set which keeps track of the context values which have been generated within the KDF oracles. If the adversary tries to get a response with a context value which was generated within the oracle, then the oracle should return an error.

$\underline{\texttt{Gind-wkdf}^0_{\mathsf{WBHW}}}$

Parameters

λ:	security parameter
WBHW:	WKDF scheme

Package State

$k_{\texttt{kdf}}$:	KDF key
$k_{\texttt{HWm}}$:	hardware main key
$k_{\texttt{HWs}}$:	hardware sub-key
Q:	context set

GETWB($label$)

assert $\texttt{WKDF} \neq \bot$
$k_{\texttt{kdf}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$k_{\texttt{HWm}} \leftarrow\!\!\$\ \mathsf{kgen}_{\texttt{HW}}(1^\lambda)$
$k_{\texttt{HWs}} \leftarrow \mathsf{SubKgen}_{\texttt{HW}}(k_{\texttt{HWm}}, label)$
$\texttt{WKDF} \leftarrow\!\!\$\ \mathsf{Comp}(k_{\texttt{kdf}}, k_{\texttt{HWs}})$
return WKDF

KDF()

$e \leftarrow\!\!\$\ \{0,1\}^\lambda$
$Q \leftarrow Q \cup \{e\}$
$\hat{k} \leftarrow \mathsf{KDF}(k_{\texttt{kdf}}, e)$
return $(\hat{k}, e)$

Resp(e, $label$)

assert $e \notin Q$
$Q \leftarrow Q \cup \{e\}$
$\sigma \leftarrow \texttt{Resp}_{\texttt{HW}}(k_{\texttt{HWm}}, label, e)$
return σ

$\underline{\texttt{Gind-wkdf}^1_{\mathsf{WBHW}}}$

Parameters

λ:	security parameter
WBHW:	WKDF scheme

Package State

$k_{\texttt{kdf}}$:	KDF key
$k_{\texttt{HWm}}$:	hardware main key
$k_{\texttt{HWs}}$:	hardware sub-key
Q:	context set

GETWB($label$)

assert $\texttt{WKDF} \neq \bot$
$k_{\texttt{kdf}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$k_{\texttt{HWm}} \leftarrow\!\!\$\ \mathsf{kgen}_{\texttt{HW}}(1^\lambda)$
$k_{\texttt{HWs}} \leftarrow \mathsf{SubKgen}_{\texttt{HW}}(k_{\texttt{HWm}}, label)$
$\texttt{WKDF} \leftarrow\!\!\$\ \mathsf{Comp}(k_{\texttt{kdf}}, k_{\texttt{HWs}})$
return WKDF

KDF()

$e \leftarrow\!\!\$\ \{0,1\}^\lambda$
$Q \leftarrow Q \cup \{e\}$
$\hat{k} \leftarrow\!\!\$\ \{0,1\}^\lambda$
return $(\hat{k}, e)$

Resp(e, $label$)

assert $e \notin Q$
$Q \leftarrow Q \cup \{e\}$
$\sigma \leftarrow \texttt{Resp}_{\texttt{HW}}(k_{\texttt{HWm}}, label, e)$
return σ