MTAT.07.003 Cryptology II
Spring 2012 / Exercise session ?? / Example Solution

Exercise (FRH in RO model). *Show that the full domain hash signature is secure against existential forgeries in the random oracle model. Establish security bounds under the assumption that the scheme uses (t, ε)-secure collection of trapdoor permutations $\mathcal{F}_{\mathrm{tp}}$ and $\mathrm{Map}_{\mathsf{pk}}(\cdot)$ is always τ-time computable.*

Solution. To show how such proofs are constructed, we construct the proof in three steps. First, we consider a very specific key only attack. Then we generalise the proof for all key only attacks. Finally we show how to convert an attack with uses signing oracle to an attack without the oracle while preserving success.

For the proof, recall that a collection of trapdoor permutations $\mathcal{F}_{\mathrm{tp}}$ is determined by three algorithms $(\mathsf{Gen}, \mathrm{Map}, \mathrm{Inv})$ such that

$$\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}, \quad \forall m \in \mathcal{M}_{\mathsf{pk}} : \qquad \mathrm{Inv}_{\mathsf{sk}}(\mathrm{Map}_{\mathsf{pk}}(m)) = m$$

and both algorithms $\mathrm{Map}_{\mathsf{pk}}(\cdot)$ and $\mathrm{Inv}_{\mathsf{sk}}(\cdot)$ are deterministic. The security of the permutation collection is defined through the success $\mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B})$ against the following game:

$$\begin{array}{l}\mathcal{Q}^{\mathcal{B}} \\ \left[\begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen} \\ x \xleftarrow{u} \mathcal{M}_{\mathsf{pk}} \\ y \leftarrow \mathrm{Map}_{\mathsf{pk}}(x) \\ \textbf{return } [\mathcal{B}(\mathsf{pk}, y) \stackrel{?}{=} x]\end{array}\right. .\end{array}$$

A collection of trapdoor permutations $\mathcal{F}_{\mathrm{tp}}$ is (t, ε)-secure if for any t-time adversary $\mathcal{B}$ the success probability $\mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B})$ is upper bounded by ε.

A VERY SPECIFIC KEY ONLY ATTACK. Let us first consider an attack, where the adversary makes q_{h} queries to the random oracle $\mathcal{O}_{\mathrm{h}}$ to forge a signature (m, s) such that $h(m)$ is computed as the 1^{st} query to $\mathcal{O}_{\mathrm{h}}$ and all q_{h} hash queries are made on distinct inputs. We start by defining the security game $\mathcal{G}_1$, where the adversary is able to forge one more signature while having access to a random oracle $\mathcal{O}_{\mathrm{h}}$:

$$\begin{array}{l}\mathcal{G}_1 \\ \left[\begin{array}{l}h \xleftarrow{u} \mathcal{H}_{\mathrm{all}} \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen} \\ (m, s) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{h}}}(\mathsf{pk}) \\ y \leftarrow h(m) \\ \textbf{return } [\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} y]\end{array}\right. .\end{array}$$

As we consider a restricted class of adversaries that forge the signature for the first query m_1 to the hash oracle $\mathcal{O}_{\mathrm{h}}$, we can rewrite the security game in more explicit manner:

$$\begin{array}{l}\mathcal{G}_2 \\ \left[\begin{array}{l}h \xleftarrow{u} \mathcal{H}_{\mathrm{all}} \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen} \\ m_1 \leftarrow \mathcal{A}(\mathsf{pk}) \\ y \leftarrow h(m_1) \\ s \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{h}}}(y) \\ \textbf{return } [\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} y]\end{array}\right. .\end{array}$$

The game makes the first query m_1 to the oracle explicit. Formally, the adversary $\mathcal{A}$ for the game $\mathcal{G}_1$ is not compatible with the game $\mathcal{G}_2$. However, there is a straightforward wrapper construction with constant overhead that makes the conversion without changing the success probability.

For constructing the inverter for any input y, we need to trick the adversary into believing that y is the output of $h(m_1)$. For this, we replace the initial oracle with a slightly modified one which outputs y if he is queried for m_1 and $\mathcal{O}_{\mathrm{h}}(m)$ otherwise:

$$\begin{array}{l}\mathcal{O}'_{\mathrm{h}}(m)\\ \left[\begin{array}{l}\mathsf{if}\ m \stackrel{?}{=} m_1\ \mathsf{then}\ \mathbf{return}\ y\\ \mathsf{else}\ \mathbf{return}\ \mathcal{O}_{\mathrm{h}}(m)\end{array}\right.\end{array}$$

To be punctual, m_1 and y are hidden parameters of $\mathcal{O}'_{\mathrm{h}}$ which must be initialised by the inverter

$$\begin{array}{l}\mathcal{B}^{\mathcal{A},\mathcal{O}_{\mathrm{h}}}(\mathsf{pk}, y)\\ \left[\begin{array}{l}h \leftarrow \mathcal{H}_{\mathrm{all}}\\ m_1 \leftarrow \mathcal{A}(\mathsf{pk})\\ s \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{h}}}(y)\\ \mathbf{return}\ s\end{array}\right.\end{array}$$

before calling $\mathcal{O}'_{\mathrm{h}}$. Note that if we use this inverter $\mathcal{B}$ in game $\mathcal{Q}^{\mathcal{B}}$, then we get

$$\begin{array}{l}\mathcal{Q}^{\mathcal{B}}\\ \left[\begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}\\ x \leftarrow_u \mathcal{M}_{\mathsf{pk}}\\ y \leftarrow \mathrm{Map}_{\mathsf{pk}}(x)\\ h \leftarrow_u \mathcal{H}_{\mathrm{all}}\\ m_1 \leftarrow \mathcal{A}(\mathsf{pk})\\ s \leftarrow \mathcal{A}^{\mathcal{O}'_{\mathrm{h}}}(y)\\ \mathbf{return}\ [s \stackrel{?}{=} x]\ .\end{array}\right.\end{array}$$

Now note that the condition $s = x$ is equivalent to the condition that $\mathrm{Map}_{\mathsf{pk}}(s) = y$ and we can inverse the choices of m and y in $\mathcal{Q}$ by firstly taking $y \leftarrow \mathcal{M}_{\mathsf{pk}}$ and then $x \leftarrow \mathrm{Inv}_{\mathsf{sk}}(y)$ without changing the output of the game. As a result, we obtain that the game $\mathcal{Q}^{\mathcal{B}}$ is equivalent to the following game:

$$\begin{array}{l}\mathcal{G}_3\\ \left[\begin{array}{l}h \leftarrow_u \mathcal{H}_{\mathrm{all}}\\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}\\ y \leftarrow_u \mathcal{M}_{\mathsf{pk}}\\ x \leftarrow \mathrm{Inv}_{\mathsf{pk}}(m)\\ m_1 \leftarrow \mathcal{A}(\mathsf{pk})\\ s \leftarrow \mathcal{A}^{\mathcal{O}'_{\mathrm{h}}}(y)\\ \mathbf{return}\ [\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} y]\ .\end{array}\right.\end{array}$$

Now the only difference between $\mathcal{G}_2$ and $\mathcal{G}_3$ is in the way y is computed. If h is chosen uniformly from $\mathcal{H}_{\mathrm{all}}$, then $y \leftarrow h(m_1)$ is also uniform in $\mathcal{M}_{\mathsf{pk}}$ and thus the games are identical. Thus, we have formally proved

$$\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A}) = \mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}^{\mathcal{O}'_{\mathrm{h}}})\ .$$

However, the latter is not sufficient for the proof, since $\mathcal{B}$ must have an oracle access to $\mathcal{O}'_{\mathrm{h}}$ to run, whereas the security of $\mathcal{F}_{\mathrm{tp}}$ is defined in terms of algorithms that make no oracle calls.

Fortunately, we can remove the oracle $\mathcal{O}'_{\mathrm{h}}$ from the construction of $\mathcal{B}$. As $\mathcal{A}$ makes distinct calls to $\mathcal{O}_{\mathrm{h}}$ so does the $\mathcal{O}'_{\mathrm{h}}$. Consequently, we can replace all replies by randomly sampled elements, which leads to the

following simplification:

$$
\begin{array}{l}
\mathcal{O}''_{\mathrm{h}}(m) \\
\left[\begin{array}{l}
\mathsf{if}\ m \stackrel{?}{=} m_1\ \mathsf{then}\ y_* \leftarrow y \\
\mathsf{else}\ y_* \leftarrow\!\!\!\!{}_{u}\ \mathcal{M}_{\mathsf{pk}} \\
\mathbf{return}\ y_*\ .
\end{array}\right.
\end{array}
$$

Since $\mathcal{O}''_{\mathrm{h}}$ makes no calls to oracles, the algorithm $\mathcal{B}^{\mathcal{A},\mathcal{O}''_{\mathrm{h}}}$ is in the valid form. As the running time of $\mathcal{B}^{\mathcal{A},\mathcal{O}''_{\mathrm{h}}}$ is $\mathrm{O}(q_{\mathrm{h}})$ steps slower than $\mathcal{A}$, we get the desired upper bound:

$$
\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A}) = \mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}^{\mathcal{O}'_{\mathrm{h}}}) = \mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}^{\mathcal{O}''_{\mathrm{h}}}) \leq \varepsilon
$$

whenever the running time of $\mathcal{A}$ is $t - \mathrm{O}(q)$.

Similar construction could be given for another restricted case when $h(m)$ is the i^{th} query. We modify the oracle such that it returns random value for all j^{th} query for $j \neq i$ and outputs y when $j = i$:

$$
\begin{array}{l}
\mathcal{O}''_{\mathrm{h}}(m) \\
\left[\begin{array}{l}
\mathit{count} \leftarrow \mathit{count} + 1 \\
\mathsf{if}\ \mathit{count} \stackrel{?}{=} i\ \mathsf{then}\ y_* \leftarrow y \\
\mathsf{else}\ y_* \leftarrow\!\!\!\!{}_{u}\ \mathcal{M}_{\mathsf{pk}} \\
\mathbf{return}\ y_*
\end{array}\right.
\end{array}
$$

where the global variable *count* is initially set to zero. Further details are left as an exercise.

A constructive reduction for a key only attack. Next, let us consider a general attack where the adversary makes q_{h} queries to the random oracle $\mathcal{O}_{\mathrm{h}}$ to forge a signature (m, s). W.l.o.g. we can assume that $\mathcal{A}$ queries the hash value for the forged signature. If not then we can write a wrapper that queries $h(m)$ as the last step. It can increase the number of queries to $q_{\mathrm{h}} + 1$. Secondly, we can assume that $\mathcal{A}$ makes exactly $q_{\mathrm{h}} + 1$ distinct oracle queries. Overhead of the corresponding wrapper is $\mathrm{O}(q_{\mathrm{h}} \log q_{\mathrm{h}})$. As a result, we can model the attack through the following game:

$$
\begin{array}{l}
\mathcal{G}^{\mathcal{A}} \\
\left[\begin{array}{l}
(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen} \\
h \leftarrow \mathcal{H}_{\mathrm{all}} \\
y_0 \leftarrow \perp \\
\quad \text{For}\ \ i \in \{1, \ldots, q_h + 1\}\ \ \text{do} \\
\quad \left[\begin{array}{l}
m_i \leftarrow A(y_{i-1}) \\
y_i \leftarrow h(m_i)
\end{array}\right. \\
(m, s) \leftarrow A(h_{q_h+1}) \\
\mathbf{return}\ [h(m) \stackrel{?}{=} \mathrm{Map}_{\mathsf{pk}}(s)]\ ,
\end{array}\right.
\end{array}
$$

which itself can be simplified due to the properties of the random function family $\mathcal{H}_{\text{all}}$:

$\mathcal{G}_1^{\mathcal{A}}$
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
- $y_0 \leftarrow \perp$
- For $i \in \{1, \ldots, q_h + 1\}$ do
  - $m_i \leftarrow A(y_{i-1})$
  - $y_i \leftarrow \mathcal{M}_{\mathsf{pk}}$
- $(m, s) \leftarrow A(h_{q_h+1})$
- Choose k such that $m = m_k$
- **return** $[h_k = \mathrm{Map}_{\mathsf{pk}}(s)]$

without changing the success probability nor the semantics of the game.

Now it is straightforward to construct the inverter $\mathcal{B}$ by planting the value y to be inverted as the k^{th} reply. Clearly for some k, the probability that the adversary $\mathcal{A}$ is successful and m is submitted as the k^{th} is not smaller than the average success:

$$\exists k: \quad \Pr\left[\mathcal{G}_1^{\mathcal{A}} = 1 \wedge m = m_k\right] \geq \frac{1}{q_{\text{h}} + 1} \cdot \mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A}) \ .$$

However, there is no evident way to determine k efficiently by looking at the code of $\mathcal{A}$. Hence, the only sensible alternative is to choose k randomly. The latter leads to the following inverter construction:

$\mathcal{B}^{\mathcal{A}}(\mathsf{pk}, y)$
- $k \leftarrow \{1, \ldots, q_h + 1\}$
- $y_0 \leftarrow \perp$
- For $i \in \{1, \ldots, q_h + 1\}$ do
  - $m_i \leftarrow \mathcal{A}(y_{i-1})$
  - if $i \stackrel{?}{=} k$ then $y_i \leftarrow y$ else $y_i \leftarrow \mathcal{M}_{\mathsf{pk}}$
- $(m, s) \leftarrow \mathcal{A}$
- if $m \stackrel{?}{=} m_k$ then **return** s else **return** $\perp$

There are many way how to estimate the success of the inverter. Here, we choose the most explicit way. We just inline the construction of $\mathcal{B}$ into the game $\mathcal{Q}$ and manipulate the game further without changing its semantics:

$\mathcal{Q}^{\mathcal{B}}$
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
- $x \leftarrow_u \mathcal{M}_{\mathsf{pk}}$
- $y \leftarrow \mathrm{Map}_{\mathsf{pk}}(x)$
- $k \leftarrow \{1, \ldots, q_h + 1\}$
- $y_0 \leftarrow \perp$
- For $i \in \{1, \ldots, q_h + 1\}$ do
  - $m_i \leftarrow \mathcal{A}(y_{i-1})$
  - if $i \stackrel{?}{=} k$ then $y_i \leftarrow y$ else $y_i \leftarrow \mathcal{M}_{\mathsf{pk}}$
- $(m, s) \leftarrow \mathcal{A}$
- if $m \stackrel{?}{=} m_k$ then **return** $[s \stackrel{?}{=} x]$ else **return** 0

Note that the last line captures the condition that $\mathcal{B}$ does not halt with $\perp$ only if $m = m_k$. Since $\mathrm{Map}_{\mathsf{pk}}$ is the permutation we can rewrite the input generation and output verification phase:

$\mathcal{Q}^{\mathcal{B}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$y \xleftarrow{u} \mathcal{M}_{\mathsf{pk}}$
$k \leftarrow \{1, \ldots, q_h + 1\}$
$y_0 \leftarrow \perp$
For $i \in \{1, \ldots, q_h + 1\}$ do
  $m_i \leftarrow \mathcal{A}(y_{i-1})$
  if $i \stackrel{?}{=} k$ then $y_i \leftarrow y$ else $y_i \leftarrow \mathcal{M}_{\mathsf{pk}}$
$(m, s) \leftarrow \mathcal{A}$
if $m \neq m_k$ then **return** 0
return $[\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} y]$.

Since the challenge y is chosen uniformly from $\mathcal{M}_{\mathsf{pk}}$, the difference between if and else branches in the cycle body vanishes and we can further simplify the game:

$\mathcal{Q}^{\mathcal{B}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$k \leftarrow \{1, \ldots, q_h + 1\}$
$y_0 \leftarrow \perp$
For $i \in \{1, \ldots, q_h + 1\}$ do
  $m_i \leftarrow \mathcal{A}(y_{i-1})$
  $y_i \leftarrow \mathcal{M}_{\mathsf{pk}}$
$(m, s) \leftarrow \mathcal{A}$
if $m \neq m_k$ then **return** 0
return $[\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} y_k]$,

which can be further describe in terms of random function sampling:

$\mathcal{Q}^{\mathcal{B}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$h \leftarrow \mathcal{H}_{\mathrm{all}}$
$y_0 \leftarrow \perp$
For $i \in \{1, \ldots, q_h + 1\}$ do
  $m_i \leftarrow \mathcal{A}(y_{i-1})$
  $y_i \leftarrow h(m_i)$
$(m, s) \leftarrow \mathcal{A}$
$k \leftarrow \{1, \ldots, q_h + 1\}$
if $m \neq m_k$ then **return** 0
return $[\mathrm{Map}_{\mathsf{pk}}(s) \stackrel{?}{=} h(m)]$.

As $\mathcal{A}$ is guaranteed to produce $m \in \{m_1, \ldots, m_{q_h+1}\}$ by our assumptions, we can formally derive

$$\Pr\left[\mathcal{Q}^{\mathcal{B}} = 1\right] = \sum_{i=1}^{q_h+1} \Pr\left[\mathcal{Q}^{\mathcal{B}} = 1 \wedge m = m_i\right] = \sum_{i=1}^{q_h+1} \Pr\left[\mathrm{Map}_{\mathsf{pk}}(s) = h(m) \wedge m = m_k \wedge m = m_i\right] \quad .$$

As the term under the sum can be expressed as

$$\Pr\left[\mathrm{Map}_{\mathsf{pk}}(s) = h(m) \wedge m = m_k \wedge m = m_i\right] = \Pr\left[\mathrm{Map}_{\mathsf{pk}}(s) = h(m) \wedge m = m_i\right] \cdot \Pr\left[m_i = m_k\right]$$

and the condition $m_i = m_k$ holds with uniform probability

$$\forall i: \quad \Pr\left[m_i = m_k\right] = \frac{1}{q_{\mathrm{h}} + 1} \ ,$$

we get the desired lower bound

$$\mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}) = \Pr\left[\mathcal{Q}^{\mathcal{B}} = 1\right] = \frac{1}{q_{\mathrm{h}} + 1} \cdot \sum_{i=1}^{q_{\mathrm{h}}+1} \Pr\left[\mathrm{Map}_{\mathsf{pk}}(s) = h(m) \wedge m = m_i\right] = \frac{\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A})}{q_{\mathrm{h}} + 1} \ .$$

Another way to prove this bound is to reorder the checks in the game:

$\mathcal{Q}^{\mathcal{B}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$h \leftarrow \mathcal{H}_{\mathrm{all}}$
$y_0 \leftarrow \perp$
For $i \in \{1, \ldots, q_h + 1\}$ do
  $m_i \leftarrow \mathcal{A}(y_{i-1})$
  $y_i \leftarrow h(m_i)$
$(m, s) \leftarrow \mathcal{A}$
if $\mathrm{Map}_{\mathsf{pk}}(s) \neq h(m)$ then **return** 0
$k \leftarrow \{1, \ldots, q_h + 1\}$
return $[m = m_k]$.

and then note that the first part of the game is identical to $\mathcal{G}_1^{\mathcal{A}}$ and the final test succeeds with the probability

$$\forall m: \quad \Pr\left[m = m_k\right] = \frac{1}{q_{\mathrm{h}} + 1} \ .$$

To summarise, we have obtained a direct reduction with

$$\mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}) = \frac{\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A})}{q_{\mathrm{h}} + 1}$$

where the running time of $\mathcal{B}$ is $\mathrm{O}(q_{\mathrm{h}} \log q_{\mathrm{h}})$ time steps longer than the running time of $\mathcal{A}$ that is not guaranteed to query distinct inputs. Therefore, if $\mathcal{F}_{\mathrm{tp}}$ is a (t, ε)-secure collection of trapdoor permutations, the full domain hash signature is $((q_{\mathrm{h}}+1)\varepsilon, t - \mathrm{O}(q_{\mathrm{h}} \log q_{\mathrm{h}}))$-secure against key-only attacks where adversary make q_{h} queries.

A GENERAL CONSTRUCTIVE REDUCTION. Finally, let us consider a general attack where the adversary makes q_{h} queries to the random oracle $\mathcal{O}_{\mathrm{h}}$ and q_{s} queries to the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ to forge a new signature (m, s) that is not in the list of replies. Let $S[\cdot]$ denote the array of queried signatures, i.e., $S[m] \neq \perp$ means that the signature has been queried form the oracle. Then we can formalise the security game as follows:

$\mathcal{G}_1$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$h \leftarrow \mathcal{H}_{\mathrm{all}}$
$(m, s) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{h}}, \mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})$
if $S[m] \neq \perp$ **return** 0
return $[h(m) \stackrel{?}{=} \mathrm{Map}_{\mathsf{pk}}(s)]$.

As before, let q_h denote the maximal number of hash queries and q_s the maximal number of signing queries the adversary can make. Also, let $H[\cdot]$ denote array of queried hashes.

First, note that by the construction $H[m] = \mathrm{Map}_{\mathsf{pk}}(S[m])$ for all messages m that are submitted to the signing oracle. Moreover, by the properties of $\mathcal{H}_{\mathrm{all}}$ all values $H[m]$ are uniformly distributed over $\mathcal{M}_{\mathsf{pk}}$. As $\mathrm{Map}_{\mathsf{pk}}$ is a permtation, we can sample uniform values from $\mathcal{M}_{\mathsf{pk}}$ by first sampling s and then computing $\mathrm{Map}_{\mathsf{pk}}(s)$. This way we can escape the hard inversion problem. Formally, we define two new oracles

$\mathcal{O}'_{\mathrm{h}}(m)$
- if $H[m] = \bot$
  - $S[m] \xleftarrow{u} \mathcal{M}$
  - $H[m] \leftarrow \mathrm{Map}_{\mathsf{pk}}(S[m])$
- **return** $H[m]$

$\mathcal{O}'_{\mathsf{Sign}}(m)$
- if $S[m] = \bot$
  - $S[m] \xleftarrow{u} \mathcal{M}$
  - $H[m] \leftarrow \mathrm{Map}_{\mathsf{pk}}(S[m])$
- **return** $S[m]$.

For the correctness, note that the construction guarantees that elements $S[m]$ and $H[m]$ are either simultaneously undefined or defined and there can be no redefinitions of undefined variables. For the reasons explained above, the behaviour of the oracle pair $\mathcal{O}'_{\mathrm{h}}$ and $\mathcal{O}'_{\mathsf{Sign}}$ is indistinguishable form the oracle pair $\mathcal{O}_{\mathrm{h}}$ and $\mathcal{O}_{\mathsf{Sign}}$. Consequently, the game $\mathcal{G}_1$ is equivalent to the game

$\mathcal{G}_2$
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
- $(m, s) \leftarrow \mathcal{A}^{\mathcal{O}'_{\mathrm{h}}, \mathcal{O}'_{\mathsf{Sign}}}(\mathsf{pk})$
- if $S[m] \neq \bot$ **return** 0
- **return** $[h(m) \stackrel{?}{=} \mathrm{Map}_{\mathsf{pk}}(s)]$.

Note that the new game $\mathcal{G}_2$ is not equivalent to the key-only attack in the random oracle model. Although the oracle $\mathcal{O}'_{\mathrm{h}}$ is indistinguishable form the random oracle $\mathcal{O}_{\mathrm{h}}$, the construction is different.

Nevertheless, we can use the idea as before to build the inverter algorithm. Let us start from non-constructive analysis. Clearly,

$$\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A}) = \sum_{i=1}^{q_{\mathrm{h}}+q_{\mathrm{s}}+1} \Pr\left[\mathcal{G}_2^{\mathcal{A}} = 1 \wedge i^{\mathrm{th}} \text{ query corresponds to the signature}\right]$$

where $\mathcal{A}$ is such that it is quarantined to query the hash of the message m. Now there must exist k such that

$$\Pr\left[\mathcal{G}_2^{\mathcal{A}} = 1 \wedge k^{\mathrm{th}} \text{ query corresponds to the signature}\right] \geq \frac{\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A})}{q_{\mathrm{h}} + q_{\mathrm{s}} + 1} \ .$$

Now for this k, we can construct an inverter $\mathcal{B}$ by modifying the pair of oracles so that they $H[m_k] = y$. The simulation can fail only if the adversary $\mathcal{A}$ queries $\mathcal{O}'_{\mathsf{Sign}}(m_k)$ but then (m_k, s) is not a valid forgery. We leave the formal construction of the inverter and its analysis to the reader.

For the constructive reduction, we just give the construction of $\mathcal{B}$ together with modified oracles:

$\mathcal{B}^{\mathcal{O}''_{\mathrm{h}}, \mathcal{O}''_{\mathsf{Sign}}}(\mathsf{pk}, y)$
- $count \leftarrow 0$
- $k \xleftarrow{u} \{1, \ldots, q_{\mathrm{h}} + q_{\mathrm{s}} + 1\}$
- $(m, s) \leftarrow \mathcal{A}^{\mathcal{O}''_{\mathrm{h}}, \mathcal{O}''_{\mathsf{Sign}}}(\mathsf{pk})$
- if $S[m] \neq \bot$ **return** 0
- if $m \neq m_k$ then **return** 0
- else **return** s

$\mathcal{O}''_{\mathrm{h}}(m)$
- $count \leftarrow count + 1$
- if $H[m] = \bot$
  - if $count \stackrel{?}{=} k$ then
    - $H[m] \leftarrow y$
  - else
    - $S[m] \xleftarrow{u} \mathcal{M}$
    - $H[m] \leftarrow \mathrm{Map}_{\mathsf{pk}}(S[m])$
- **return** $H[m]$

$\mathcal{O}''_{\mathsf{Sign}}(m)$
- $count \leftarrow count + 1$
- if $S[m] = \bot$
  - if $count \stackrel{?}{=} k$ then
    - $H[m] \leftarrow y$
  - else
    - $S[m] \xleftarrow{u} \mathcal{M}$
    - $H[m] \leftarrow \mathrm{Map}_{\mathsf{pk}}(S[m])$
- **return** $S[m]$.

For the analysis note that $\mathcal{B}^{\mathcal{O}''_{\mathrm{h}},\mathcal{O}''_{\mathsf{Sign}}}$ provides a perfect simulation of $\mathcal{G}_2$ for $\mathcal{A}$ unless the k^{th} query is a signing query. However, the $\mathcal{B}^{\mathcal{O}''_{\mathrm{h}},\mathcal{O}''_{\mathsf{Sign}}}$ would halt even if the signature would be correct. Hence, it is straightforward to show by direct inlining that

$$\mathsf{Adv}^{\text{inv-cpa}}_{\mathcal{F}_{\mathrm{tp}}}(\mathcal{B}^{\mathcal{O}''_{\mathrm{h}},\mathcal{O}''_{\mathsf{Sign}}}) = \frac{\mathsf{Adv}^{\mathsf{win}}_{\mathcal{G}_1}(\mathcal{A})}{q_{\mathrm{h}} + q_{\mathrm{s}} + 1} \ .$$

The reduction overhead is $q_s + q_h + 1$ calls to one of $\mathcal{O}''_{\mathrm{h}}$ and $\mathcal{O}''_{\mathsf{Sign}}$, but both of these have time dominated by the $\mathrm{Map}_{\mathsf{pk}}(\cdot)$ operation, hence there is a $(q_s + q_h + 1)\tau$ overhead. So if the scheme uses (t, ε)-secure collection of trapdoor permutations $\mathcal{F}_{\mathrm{tp}}$, then the full domain hash is $(t - (q_s + q_h + 1)\tau, (q_s + q_h + 1)\varepsilon)$-secure in the random oracle model.