On the rekeying (in)separability

Kirill Tsaregorodtsev
Researcher at Cryptography laboratory,
JSRPC “Kryptonite”, Moscow, Russia

CTCrypt’2023
RUMP session

КРИПТОНИТ

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

- if rekeying function f is modelled as a random oracle, the scheme is secure;

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

- if rekeying function f is modelled as a random oracle, the scheme is secure;
- once f is instantiated with **any** efficiently computable function with "compact" representation (e.g., hash function; encrypting constants, etc.), the scheme is no longer secure in any meaningful security model;

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

- if rekeying function f is modelled as a random oracle, the scheme is secure;
- once f is instantiated with **any** efficiently computable function with “compact” representation (e.g., hash function; encrypting constants, etc.), the scheme is no longer secure in any meaningful security model;
- the interpretation of the result: even if the encryption scheme $\mathcal{SE}$ admits “idealized” internal rekeying, there is still no guarantee that it can be instantiated with **some** rekeying function;

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

- if rekeying function f is modelled as a random oracle, the scheme is secure;
- once f is instantiated with **any** efficiently computable function with "compact" representation (e.g., hash function; encrypting constants, etc.), the scheme is no longer secure in any meaningful security model;
- the interpretation of the result: even if the encryption scheme $\mathcal{SE}$ admits "idealized" internal rekeying, there is still no guarantee that it can be instantiated with **some** rekeying function;
- hence, the consideration of encryption scheme together with rekeying function is **unavoidable**,

Idealized rekeying

Main idea: it is possible to construct encryption scheme $\mathcal{SE}$ with (internal) rekeying function f with the following properties:

- if rekeying function f is modelled as a random oracle, the scheme is secure;
- once f is instantiated with **any** efficiently computable function with “compact” representation (e.g., hash function; encrypting constants, etc.), the scheme is no longer secure in any meaningful security model;
- the interpretation of the result: even if the encryption scheme $\mathcal{SE}$ admits “idealized” internal rekeying, there is still no guarantee that it can be instantiated with **some** rekeying function;
- hence, the consideration of encryption scheme together with rekeying function is **unavoidable**,
- (internal) rekeying is the **inseparable** part of the encryption scheme.

Injecting weakness

- Let $\widehat{\mathcal{SE}}$ be a "good" encryption scheme that admits idealized rekeying; let us screw it up and obtain a "bad" scheme $\mathcal{SE}$.

Injecting weakness

- Let $\widehat{\mathcal{SE}}$ be a "good" encryption scheme that admits idealized rekeying; let us screw it up and obtain a "bad" scheme $\mathcal{SE}$.
- $\mathcal{SE}$ works as follows: on the input m it firstly runs distinguishing algorithm $\mathcal{D}(f, m)$:

Injecting weakness

- Let $\widehat{\mathcal{SE}}$ be a "good" encryption scheme that admits idealized rekeying; let us screw it up and obtain a "bad" scheme $\mathcal{SE}$.
- $\mathcal{SE}$ works as follows: on the input m it firstly runs distinguishing algorithm $\mathcal{D}(f, m)$:
  - if $\mathcal{D}$ returns 0, a string of 0's is appended to the end of the ciphertext,

Injecting weakness

- Let $\widehat{\mathcal{SE}}$ be a "good" encryption scheme that admits idealized rekeying; let us screw it up and obtain a "bad" scheme $\mathcal{SE}$.
- $\mathcal{SE}$ works as follows: on the input m it firstly runs distinguishing algorithm $\mathcal{D}(f, m)$:
  - if $\mathcal{D}$ returns 0, a string of 0's is appended to the end of the ciphertext,
  - if $\mathcal{D}$ returns 1, the secret key K is appended to the end of the ciphertext.

Injecting weakness

- Let $\widehat{\mathcal{SE}}$ be a "good" encryption scheme that admits idealized rekeying; let us screw it up and obtain a "bad" scheme $\mathcal{SE}$.
- $\mathcal{SE}$ works as follows: on the input m it firstly runs distinguishing algorithm $\mathcal{D}(f, m)$:
  - if $\mathcal{D}$ returns 0, a string of 0's is appended to the end of the ciphertext,
  - if $\mathcal{D}$ returns 1, the secret key K is appended to the end of the ciphertext.
- The resulting ciphertext consists of $ct \leftarrow \widehat{\mathcal{SE}}.\mathsf{Enc}(K, m)$ and 0^{klen} / K.

Distinguisher

$\mathcal{D}(f, m)$:

- m is interpreted as **a pair** (π, t), where π is some program for the Universal Turing Machine, t is the maximal number of steps;

Distinguisher

$\mathcal{D}(f, m)$:

- m is interpreted as **a pair** (π, t), where π is some program for the Universal Turing Machine, t is the maximal number of steps;
- $\mathcal{D}$ chooses $x \xleftarrow{\mathcal{U}} \{0,1\}^{klen}$, runs π on input x no more than t steps and obtains y;

Distinguisher

$\mathcal{D}(f, m)$:

- m is interpreted as **a pair** (π, t), where π is some program for the Universal Turing Machine, t is the maximal number of steps;
- $\mathcal{D}$ chooses $x \xleftarrow{\mathcal{U}} \{0,1\}^{klen}$, runs π on input x no more than t steps and obtains y;
- then $\mathcal{D}$ queries $f(x)$ and obtains y';

Distinguisher

$\mathcal{D}(f, m)$:

- m is interpreted as **a pair** (π, t), where π is some program for the Universal Turing Machine, t is the maximal number of steps;
- $\mathcal{D}$ chooses $x \overset{\mathcal{U}}{\longleftarrow} \{0,1\}^{klen}$, runs π on input x no more than t steps and obtains y;
- then $\mathcal{D}$ queries $f(x)$ and obtains y';
- if $y = y'$, then $\mathcal{D}$ returns 1, otherwise 0.

Main idea: there is no "compact" program π on the UTM for the Random Oracle.

What can be done then?

- We still want modularity of the analysis, is this possible?

What can be done then?

- We still want modularity of the analysis, is this possible?
- Yes, in the following sense: we can consider two separate sub-problems:

What can be done then?

- We still want modularity of the analysis, is this possible?
- Yes, in the following sense: we can consider two separate sub-problems:
  - How $\mathcal{SE}$ behaves if is used with d independent keys (instead of a "rekeyed" ones)?

What can be done then?

- We still want modularity of the analysis, is this possible?
- Yes, in the following sense: we can consider two separate sub-problems:
  - How $\mathcal{SE}$ behaves if is used with d independent keys (instead of a "rekeyed" ones)?
  - How $\mathcal{SE}$ behaves in **one round** of rekeying?

Leakage/Extractor

Let $\texttt{Leak}$ be a function dependent on the secret key K; $\texttt{Ext}$ is some function (extractor). Define the following Ext model.

Leakage/Extractor

Let **Leak** be a function dependent on the secret key K; **Ext** is some function (extractor). Define the following Ext model.

$$\mathsf{Adv}^{\mathrm{Ext}}_{\mathcal{SE}}(\mathcal{A}) = \mathbb{P}\Big[\mathbf{Exp}^{\mathrm{Ext}\text{-}1}_{\mathbf{Ext}}(\mathcal{A}) \to 1\Big] - \mathbb{P}\Big[\mathbf{Exp}^{\mathrm{Ext}\text{-}0}_{\mathbf{Ext}}(\mathcal{A}) \to 1\Big],$$

Leakage/Extractor

Let **Leak** be a function dependent on the secret key K; **Ext** is some function (extractor). Define the following Ext model.

$$\mathsf{Adv}_{\mathcal{SE}}^{\mathrm{Ext}}(\mathcal{A}) = \mathbb{P}\Big[\mathbf{Exp}_{\mathsf{Ext}}^{\mathrm{Ext\text{-}1}}(\mathcal{A}) \to 1\Big] - \mathbb{P}\Big[\mathbf{Exp}_{\mathsf{Ext}}^{\mathrm{Ext\text{-}0}}(\mathcal{A}) \to 1\Big],$$

$\mathbf{Exp}_{\mathsf{Ext}}^{\mathrm{Ext\text{-}}b}(\mathcal{A})$	$\mathcal{O}^{\mathrm{leak}}(m)$
$K \xleftarrow{\$} \mathsf{KGen}$	**return** $\mathsf{Leak}_K(m)$
if $b = 0$	
$\quad K' \xleftarrow{\$} \mathsf{KGen}$	
else	
$\quad K' \leftarrow \mathsf{Ext}(K)$	
fi	
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathrm{leak}}}(K')$	
return b'	

Extractor

- Ext formalized the following logic: even if the adversary has an access to the "additional information" about the key K (via $\mathtt{Leak}_K$), it cannot distinguish the "real" rekeyed value $K' \leftarrow \mathtt{Ext}(K)$ from the random one $K' \xleftarrow{\$} \mathsf{KGen}$.

Extractor

- Ext formalized the following logic: even if the adversary has an access to the "additional information" about the key K (via $\texttt{Leak}_K$), it cannot distinguish the "real" rekeyed value $K' \leftarrow \texttt{Ext}(K)$ from the random one $K' \xleftarrow{\$} \mathsf{KGen}$.
- This model can be used to partially separate rekeying process from the encryption process (but not completely! due to the $\texttt{Leak}_K$).

Thank you for your attention!

Author(s):

Tsaregorodtsev Kirill

Researcher at Cryptography laboratory,
JSRPC “Kryptonite”, Moscow, Russia
k.tsaregorodtsev@kryptonite.ru