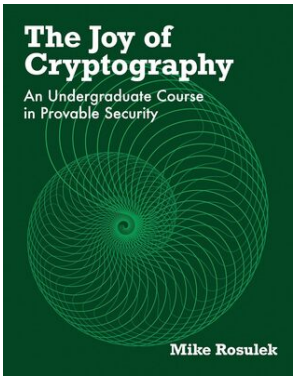
The Joy of Library-Based Security

Mike Rosulek

Oregon State University OSU

May 8: ProTeCS workshop @ Eurocrypt 2026



personal reflections on writing a textbook about *provable security*

research

expert audience

advanced proof techniques

teaching

novice audience

fundamental results only

a decade in the making

2012 frustration

…with available textbook options

a decade in the making

2012 **frustration**

…with available textbook options

2015 **enlightenment**

"I think I could present OTP, PRG, PRF, CPA, in a unified way…"

a decade in the making

2012 **frustration**

…with available textbook options

2015 **enlightenment**

"I think I could present OTP, PRG, PRF, CPA, in a unified way…"

2018 **excitement**

The Joy of Cryptography project becomes public

a decade in the making

2012 **frustration**
…with available textbook options

2015 **enlightenment**
"I think I could present OTP, PRG, PRF, CPA, in a unified way…"

2018 **excitement**
The Joy of Cryptography project becomes public

2023–4 **endless drudgery**
publisher contract, complete rewrite from scratch

a decade in the making

2012 **frustration**
…with available textbook options

2015 **enlightenment**
"I think I could present OTP, PRG, PRF, CPA, in a unified way…"

2018 **excitement**
The Joy of Cryptography project becomes public

2023–4 **endless drudgery**
publisher contract, complete rewrite from scratch

2026 **relief**
appears in print January 6

2012 *frustration*

2 games?

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

$$\Pr_{K}[\mathcal{A}^{F(K,\cdot)}() = 1] \approx \Pr_{R}[\mathcal{A}^{R}() = 1]$$

2 games?

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

$$\Pr_{K}[\mathcal{A}^{F(K,\cdot)}() = 1] \approx \Pr_{R}[\mathcal{A}^{R}() = 1]$$

1 game?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

2 games?

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

$$\Pr_{K}[\mathcal{A}^{F(K,\cdot)}() = 1] \approx \Pr_{R}[\mathcal{A}^{R}() = 1]$$

1 game?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

whatever this is

$$\Pr[M = m] = \Pr[M = m \mid C = c]$$

adversary gets input

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

adversary gets input

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

adversary has oracle

$$\Pr_{K}[\mathcal{A}^{F(K,\cdot)}() = 1] \approx \Pr_{R}[\mathcal{A}^{R}() = 1]$$

adversary gets input

$$\Pr_{S}[\mathcal{A}(G(S)) = 1] \approx \Pr[\mathcal{A}(U_{\lambda+\ell}) = 1]$$

adversary has oracle

$$\Pr_{K}[\mathcal{A}^{F(K,\cdot)}() = 1] \approx \Pr_{R}[\mathcal{A}^{R}() = 1]$$

both, in different phases

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

guess the secret bit?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

artificial “definition smell”

guess the secret bit?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

artificial “definition smell”

generate forgery?

The signature experiment $\mathsf{Sig\text{-}forge}^{\mathsf{cma}}_{\mathcal{A},\Pi}(n)$**.**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and oracle access to* $\mathsf{Sign}_{sk}(\cdot)$. *(This oracle returns a signature* $\mathsf{Sign}_{sk}(m)$ *for any message* m *of the adversary's choice.) The adversary then outputs* (m, σ).
3. *Let* $\mathcal{Q}$ *denote the set of messages whose signatures were requested by* $\mathcal{A}$ *during its execution. The output of the experiment is 1 if* $m \notin \mathcal{Q}$ *and* $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

interaction as prose?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

interaction as prose?

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

interaction as "code"?

$$(pk, sk) \twoheadleftarrow \mathsf{Gen}(1^n)$$

$$(m_0, m_1, state) := \mathcal{A}^{\mathsf{Enc}_{pk}(\cdot)}(pk)$$

assert $|m_0| = |m_1|$

$b \twoheadleftarrow \{0, 1\}$

$c := \mathsf{Enc}_{pk}(m_b)$

$$b' := \mathcal{A}^{\mathsf{Enc}_{pk}(\cdot)}(state, c)$$

return $[b \stackrel{?}{=} b']$

primitive interface ≠ security game interface

The CPA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *as well as oracle access to* $\mathsf{Enc}_{pk}(\cdot)$. *The adversary outputs a pair of messages* m_0, m_1 *with* $|m_0| = |m_1|$. *(These messages must be in the plaintext space associated with* pk*.)*
3. *A random bit* $b \in \{0,1\}$ *is chosen, and then the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$. *We call* c *the* challenge ciphertext. $\mathcal{A}$ *continues to have access to* $\mathsf{Enc}_{pk}(\cdot)$.
4. $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be 1 if* $b' = b$, *and 0 otherwise.*

</nitpick>

2015 *enlightenment*

2015 *enlightenment*

organizing principles of library-based security

start with

1 game-based security

[Shoup04, BellareRogaway06]

start with

1 game-based security

[Shoup04, BellareRogaway06]

write every security definition as

2 indistinguishability of two separate games

death to "guess the secret bit!"

1 start with
game-based security
[Shoup04, BellareRogaway06]

2 write every security definition as
indistinguishability of two separate games

death to "guess the secret bit!"

3 # reductions are implicit
in the game-hopping flow

an example

1: ~~game-based~~ *library*-based security

library = stateful collection of subroutines

$K \twoheadleftarrow \{0,1\}^\lambda$

ENC(M):
  return Enc(K, M)

DEC(C):
  return Dec(K, C)

QUERY(X):
  if $L[X]$ undefined:
    $L[X] \twoheadleftarrow \{0,1\}^n$
  return $L[X]$

1: ~~game-based~~ library-based security

library = stateful collection of subroutines

$K \twoheadleftarrow \{0,1\}^\lambda$

ENC(M):
  return Enc(K, M)

DEC(C):
  return Dec(K, C)

QUERY(X):
  if $L[X]$ undefined:
    $L[X] \twoheadleftarrow \{0,1\}^n$
  return $L[X]$

"$\mathcal{A} \diamond \mathcal{L}$" = result of composing (calling program) $\mathcal{A}$ with (library) $\mathcal{L}$ in the natural way

1: ~~game-based~~ library-based security

library = stateful collection of subroutines

$K \twoheadleftarrow \{0,1\}^\lambda$

ENC(M):
  return Enc(K, M)

DEC(C):
  return Dec(K, C)

QUERY(X):
  if $L[X]$ undefined:
    $L[X] \twoheadleftarrow \{0,1\}^n$
  return $L[X]$

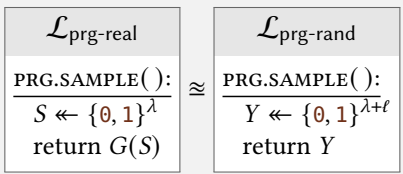
"$\mathcal{A} \diamond \mathcal{L}$" = result of composing (calling program) $\mathcal{A}$ with (library) $\mathcal{L}$ in the natural way

only interaction between adversary & "challenger" is subroutine-calling
(variables are *privately* scoped to the library)

2: security = indistinguishability of 2 libraries

Definition

$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ is a **secure pseudorandom generator** if:



2: security = indistinguishability of 2 libraries

Definition

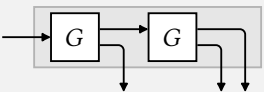
$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ is a **secure pseudorandom generator** if:

$\mathcal{L}_{\text{prg-real}}$		$\mathcal{L}_{\text{prg-rand}}$
PRG.SAMPLE( ):	$\approx$	PRG.SAMPLE( ):
$S \twoheadleftarrow \{0,1\}^\lambda$		$Y \twoheadleftarrow \{0,1\}^{\lambda+\ell}$
return $G(S)$		return Y

For all poly-time $\mathcal{A}$: $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{prg-real}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{prg-rand}} \Rightarrow 1]$

security proofs: an example

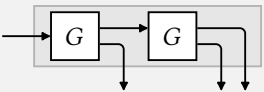
Claim:

If G is a secure (length-doubling) PRG then 

is a secure PRG

security proofs: an example

Claim:

If G is a secure (length-doubling) PRG then 

is a secure PRG

if

$\mathcal{L}^{G}_{\text{prg-real}}$

$\textsc{prg.sample}_G()$:
$S \leftarrow \{0,1\}^{\lambda}$
return $G(S)$

$\approx$

$\mathcal{L}^{G}_{\text{prg-rand}}$

$\textsc{prg.sample}_G()$:
$Y \leftarrow \{0,1\}^{2\lambda}$
return Y

then

$\mathcal{L}^{H}_{\text{prg-real}}$

$\textsc{prg.sample}_H()$:
$S \leftarrow \{0,1\}^{\lambda}$
$A \| B := G(S)$
$C \| D := G(B)$
return $A \| C \| D$

$\approx$

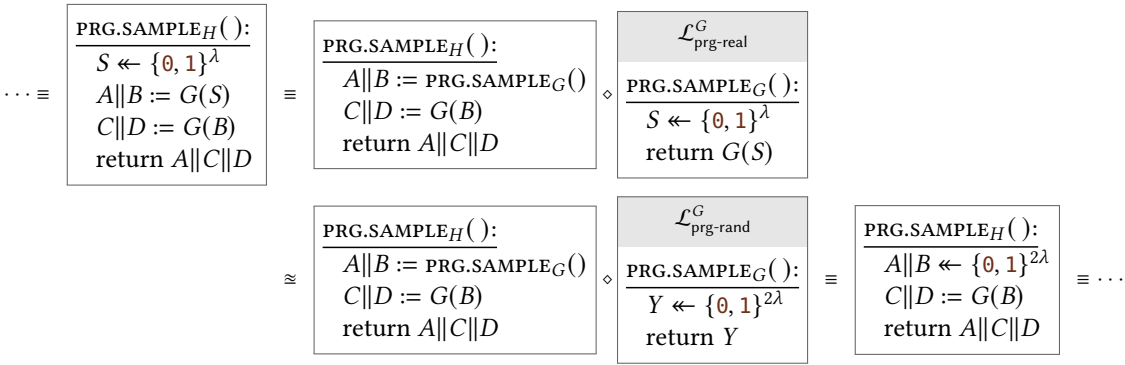
$\mathcal{L}^{H}_{\text{prg-rand}}$

$\textsc{prg.sample}_H()$:
$Y \leftarrow \{0,1\}^{3\lambda}$
return Y

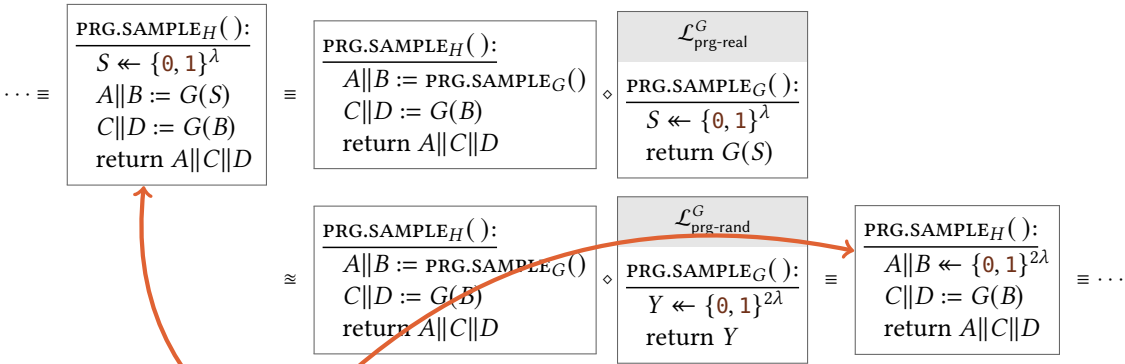
<demo>

where was the reduction?

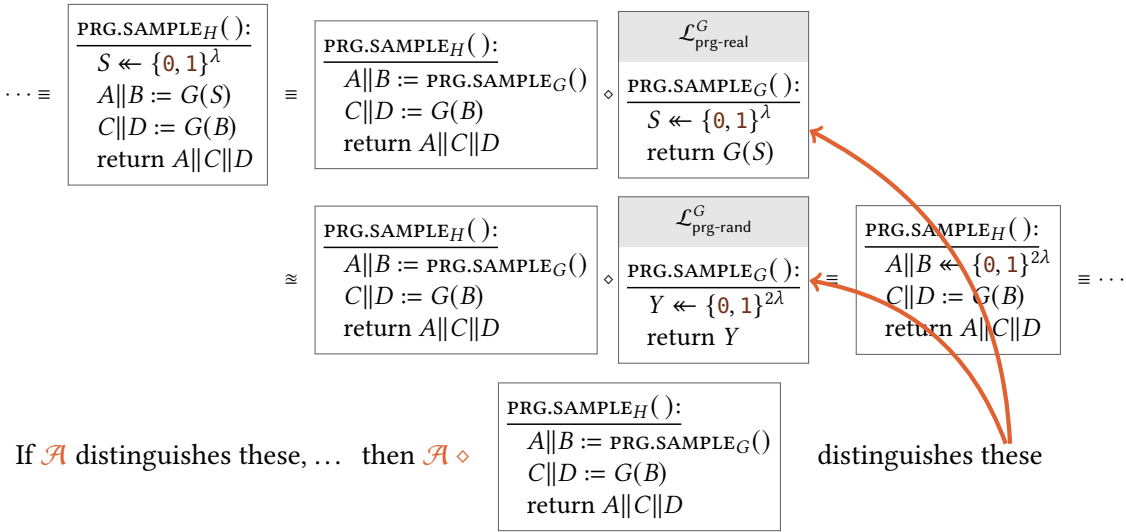
where was the reduction?



where was the reduction?



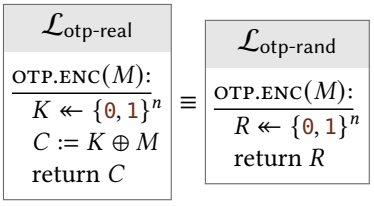
where was the reduction?



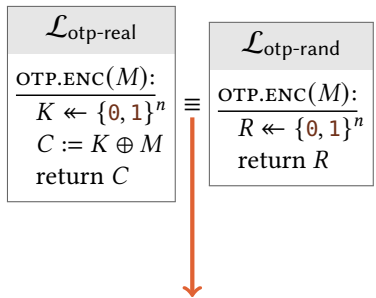
more detail:

a tour of security definitions

One-time-pad security property:

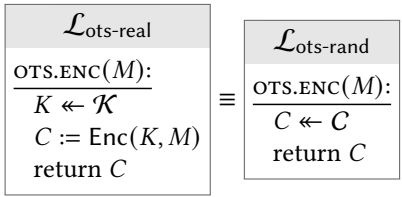


One-time-pad security property:



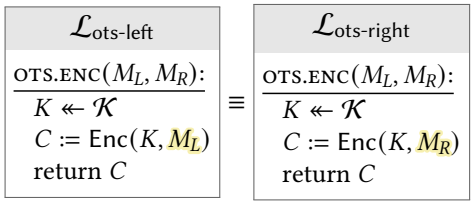
For all (not necessarily poly-time) $\mathcal{A}$: $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{otp-real}} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{otp-rand}} \Rightarrow 1]$

One-time secrecy for symmetric-key encryption:

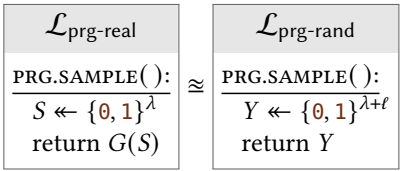


(real-or-random!)

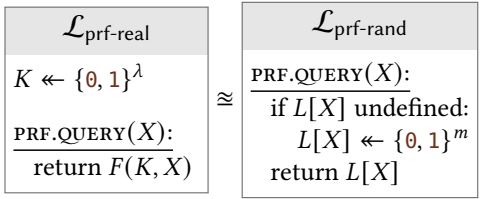
Left-or-right one-time secrecy for symmetric-key encryption:



$G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ is a **secure pseudorandom generator** if:

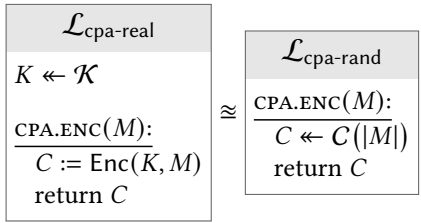


Security of a **pseudorandom function**:



("lazy random dictionary")

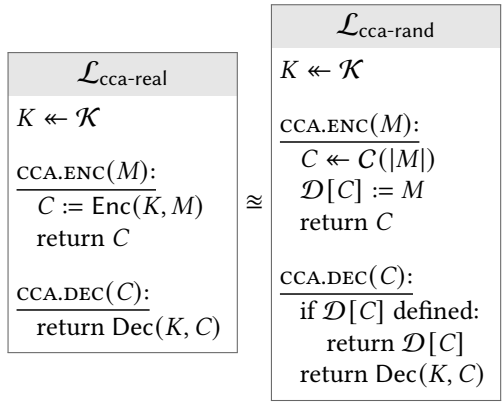
CPA security for symmetric-key encryption:



$\mathcal{C}(\ell)$ = set of ciphertexts that result from encrypting plaintexts of length ℓ

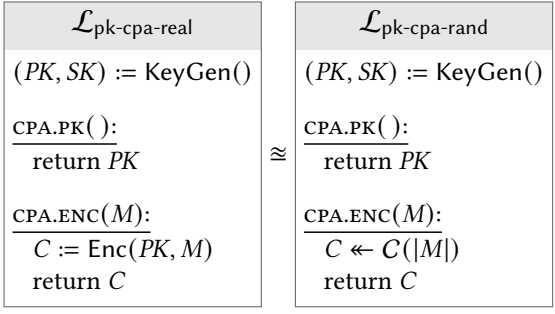
every ciphertext is a "challenge" ciphertext

CCA security for symmetric-key encryption:

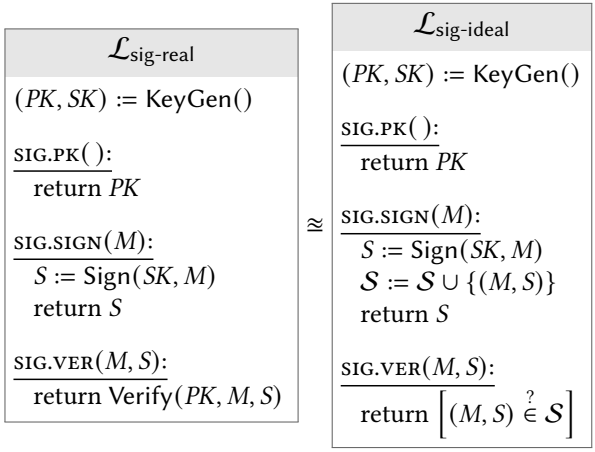


"patch" the decryption oracle vs "guard" the decryption oracle

CPA security for public-key encryption:



Security for a **digital signature scheme**:

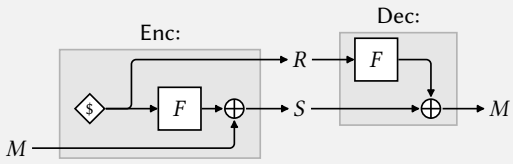


every call to SIG.VER is a potential for forgery

a tour of some security proofs

Claim:

If F is a secure PRF with input length λ, then the following is a CPA-secure symmetric encryption scheme:



$\underline{\mathsf{Enc}(K, M)}$:
$R \twoheadleftarrow \{0,1\}^\lambda$
$S := F(K, R) \oplus M$
return $R \| S$

in other words…

if

$\mathcal{L}_{\text{prf-real}}$

$K \twoheadleftarrow \{0,1\}^\lambda$

PRF.QUERY(X):

return $F(K, X)$

$\approx$

$\mathcal{L}_{\text{prf-rand}}$

PRF.QUERY(X):

if $L[X]$ undefined:

$L[X] \twoheadleftarrow \{0,1\}^n$

return $L[X]$

then

$\mathcal{L}_{\text{cpa-real}}$

$K \twoheadleftarrow \{0,1\}^\lambda$

CPA.ENC(M):

$R \twoheadleftarrow \{0,1\}^\lambda$

$S := F(K, R) \oplus M$

return $R \| S$

$\approx$

$\mathcal{L}_{\text{cpa-rand}}$

CPA.ENC(M):

$R \| S \twoheadleftarrow \{0,1\}^{\lambda+n}$

return $R \| S$

<demo>

idiom: 3-hop maneuver

$\mathcal{L}_1 \equiv \mathcal{L}_2 \diamond \mathcal{L}_{\text{left}}$ *(factor out)*

idiom: 3-hop maneuver

$$\mathcal{L}_1 \equiv \mathcal{L}_2 \diamond \mathcal{L}_{\text{left}} \qquad \textit{(factor out)}$$
$$\approx \mathcal{L}_2 \diamond \mathcal{L}_{\text{right}} \qquad \textit{(swap)}$$

$\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$ defines
security of some primitive

idiom: 3-hop maneuver

$$
\begin{aligned}
\mathcal{L}_1 &\equiv \mathcal{L}_2 \diamond \mathcal{L}_{\text{left}} && \textit{(factor out)} \\
&\approx \mathcal{L}_2 \diamond \mathcal{L}_{\text{right}} && \textit{(swap)} \\
&\equiv \mathcal{L}_3 && \textit{(inline)}
\end{aligned}
$$

$\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$ defines
security of some primitive

idiom: 3-hop maneuver

$$\begin{aligned} \mathcal{L}_1 &\equiv \mathcal{L}_2 \diamond \mathcal{L}_{\text{left}} && \textit{(factor out)} \\ &\approx \mathcal{L}_2 \diamond \mathcal{L}_{\text{right}} && \textit{(swap)} \\ &\equiv \mathcal{L}_3 && \textit{(inline)} \end{aligned}$$

$\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$ defines security of some primitive

A reduction in disguise:

If $\mathcal{A}$ distinguishes $\mathcal{L}_1$ from $\mathcal{L}_3$ then $\mathcal{A} \diamond \mathcal{L}_2$ breaks the primitive

if a proof fails, it's in the *factoring out* step

another example

idiom: bad events

Theorem

If $\mathcal{L}_1, \mathcal{L}_2$ identical-until-bad, then advantage is bounded by

$$\Pr[\text{bad event triggered in } \mathcal{A} \diamond \mathcal{L}_1]$$

idiom: bad events

Theorem

If $\mathcal{L}_1, \mathcal{L}_2$ identical-until-bad, then advantage is bounded by

$$\Pr[\text{bad event triggered in } \mathcal{A} \diamond \mathcal{L}_1]$$

problem: some subtlety in analyzing bad event probability

2-block CBC-MAC:

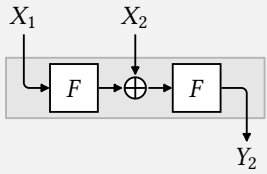
If F is a secure PRF with input/output length λ, then the following a secure PRF with input length 2λ:

$F^2_{\text{cbc}}(K, X_1 \| X_2)$:

$Y_1 := F(K, X_1)$

$Y_2 := F(K, Y_1 \oplus X_2)$

return Y_2



in other words…

if

$\mathcal{L}_{\text{prf-real}}$

$K \twoheadleftarrow \{0,1\}^{\lambda}$

$\underline{\textsc{prf.query}(X):}$
return $F(K, X)$

$\approx$

$\mathcal{L}_{\text{prf-rand}}$

$\underline{\textsc{prf.query}(X):}$
if $L[X]$ undefined:
$L[X] \twoheadleftarrow \{0,1\}^{n}$
return $L[X]$

then

$\mathcal{L}^{\text{cbc-2}}_{\text{prf-real}}$

$K \twoheadleftarrow \{0,1\}^{\lambda}$

$\underline{\textsc{prf.query}_{\text{cbc-2}}(X_1 \| X_2):}$
$Y_1 := F(K, X_1)$
$Y_2 := F(K, Y_1 \oplus X_2)$
return Y_2

$\approx$

$\mathcal{L}^{\text{cbc-2}}_{\text{prf-rand}}$

$\underline{\textsc{prf.query}_{\text{cbc-2}}(X_1 \| X_2):}$
if $L[X_1 \| X_2]$ undefined:
$L[X_1 \| X_2] \twoheadleftarrow \{0,1\}^{\lambda}$
return $L[X_1 \| X_2]$

<demo>

$\mathcal{L}_{\text{prf-rand}}$

PRF.QUERY$(X_1 \| X_2)$:

if $L[X_1 \| X_2]$ undefined:

$L[X_1 \| X_2] \twoheadleftarrow \{0,1\}^\lambda$

$\mathcal{X} := \mathcal{X} \cup \{X_1 \| X_2\}$

return $L[X_1 \| X_2]$

END OF TIME:

for $X_1 \| X_2 \in \mathcal{X}$:

if $R[X_1]$ undefined: $R[X_1] \twoheadleftarrow \{0,1\}^\lambda$

$Y_1 := R[X_1]$

if $R[Y_1 \oplus X_2]$ defined:

bad := true

$R[Y_1 \oplus X_2] := L[X_1 \| X_2]$

What is the bad event probability?

$\mathcal{L}_{\text{prf-rand}}$

PRF.QUERY($X_1 \| X_2$):
  if $L[X_1 \| X_2]$ undefined:
    $L[X_1 \| X_2] \twoheadleftarrow \{0,1\}^\lambda$
    $\mathcal{X} := \mathcal{X} \cup \{X_1 \| X_2\}$
  return $L[X_1 \| X_2]$

END OF TIME:
  for $X_1 \| X_2 \in \mathcal{X}$:
    if $R[X_1]$ undefined: $R[X_1] \twoheadleftarrow \{0,1\}^\lambda$
    $Y_1 := R[X_1]$
    if $R[Y_1 \oplus X_2]$ defined:
      bad := true
    $R[Y_1 \oplus X_2] := L[X_1 \| X_2]$

What is the bad event probability?

all randomness chosen after adversary's inputs ($\mathcal{X}$) are chosen

$\Rightarrow$ fairly straight-forward union bound

philosophical musings

✗

contrapositive reasoning

if $\mathcal{A}$ breaks Σ then $\mathcal{B}$ breaks primitive

✓

positive reasoning

if primitive secure then Σ secure

✗

contrapositive reasoning

if $\mathcal{A}$ breaks Σ then $\mathcal{B}$ breaks primitive

focus is on unspecified, hypothetical, mysterious $\mathcal{A}$

✓

positive reasoning

if primitive secure then Σ secure

focus is on *tangible* libraries that define security for Σ

contrapositive reasoning

if $\mathcal{A}$ breaks Σ then $\mathcal{B}$ breaks primitive

focus is on unspecified, hypothetical, mysterious $\mathcal{A}$

$\mathcal{B}$ appears out of thin air,
one big conceptual leap

positive reasoning

if primitive secure then Σ secure

focus is on *tangible* libraries that define security for Σ

cryptographic reasoning

CS101 reasoning

cryptographic reasoning

probabilities, distributions, views

CS101 reasoning

behavior of concrete source code
(CS major's native language)

cryptographic reasoning

probabilities, distributions, views

indistinguishability

CS101 reasoning

behavior of concrete source code
(CS major's native language)

these programs "do the same thing"

cryptographic reasoning

probabilities, distributions, views

indistinguishability

secrecy of values

CS101 reasoning

behavior of concrete source code
(CS major's native language)

these programs "do the same thing"

lexical scope

cryptographic reasoning

probabilities, distributions, views

indistinguishability

secrecy of values

independence of random variables

CS101 reasoning

behavior of concrete source code
(CS major's native language)

these programs "do the same thing"

lexical scope

syntactic independence of variables
("can sample distribution *afterwards*")

difficulties, limitations

awkward things

security definitions for CRHF, OWF

awkward things

security definitions for CRHF, OWF

so much *"bookkeeping!"*

- multi-stage games
- guarding an oracle
- adversary repeating queries

awkward things

security definitions for CRHF, OWF

so much *"bookkeeping!"*

- multi-stage games
- guarding an oracle
- adversary repeating queries

single-instance vs multi-instance

$\mathcal{L}_{\text{prg-real}}$

PRG.SAMPLE( ):
$S \twoheadleftarrow \{0,1\}^\lambda$
return $G(S)$

vs

$\mathcal{L}^F_{\text{prf-real}}$

$K \twoheadleftarrow \{0,1\}^\lambda$

PRF.QUERY(X):
return $F(K, X)$

awkward things

security definitions for CRHF, OWF

so much *"bookkeeping!"*

- multi-stage games
- guarding an oracle
- adversary repeating queries

single-instance vs multi-instance

$\mathcal{L}_{\text{prg-real}}$

$\underline{\text{PRG.SAMPLE}()}$:
$S \twoheadleftarrow \{0,1\}^\lambda$
return $G(S)$

vs

$\mathcal{L}^F_{\text{prf-real}}$

$K \twoheadleftarrow \{0,1\}^\lambda$

$\underline{\text{PRF.QUERY}(X)}$:
return $F(K, X)$

proof techniques:

- reduction rewinds the adversary (forking lemma)
- reduction "guesses the oracle query"
- …

but can you use it for research?

State Separation for Code-Based Game-Playing Proofs

Chris Brzuska[1], Antoine Delignat-Lavaud[2], Cédric Fournet[2], Konrad Kohbrok[1], and Markulf Kohlweiss[2,3]

ia.cr/2018/306

personal reflections

security = indistinguishability of
2 separate games

personal reflections

security = indistinguishability of
2 separate games

"3-hop maneuver mindset"

personal reflections

security = indistinguishability of
2 separate games

"3-hop maneuver mindset"

(MPC:) real world and ideal world are just
interactions with an adversary

personal reflections

security = indistinguishability of 2 separate games

"3-hop maneuver mindset"

(MPC:) real world and ideal world are just interactions with an adversary

I have more confidence about details (of definitions, protocols, ideal functionalities)

personal reflections

security = indistinguishability of 2 separate games

"3-hop maneuver mindset"

(MPC:) real world and ideal world are just interactions with an adversary

I have more confidence about details (of definitions, protocols, ideal functionalities)

MPC interactions are complicated, many distinct *phases*

personal reflections

✓

security = indistinguishability of 2 separate games

"3-hop maneuver mindset"

(MPC:) real world and ideal world are just interactions with an adversary

I have more confidence about details (of definitions, protocols, ideal functionalities)

✗

MPC interactions are complicated, many distinct *phases*

reviewers not yet sufficiently indoctrinated

case study from a recent submission

an interactive 2PC protocol:

Alice (input X, $\lvert X\rvert = n_A$):		Bob (input Y, $\lvert Y\rvert = n_B$):
for $x \in X$: $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$ $S_A = S_A \cup \{(x, pk_x)\}$ $D_A = \mathsf{Encode}_A^{O_A}(S_A)$	$\xrightarrow{D_A}$	for $y \in Y$: $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$ $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$ $S_B = S_B \cup \{(y, C_y)\}$ $D_B = \mathsf{Encode}_B^{O_B}(S_B)$
for $x \in X$: $\widetilde{C}_x = \mathsf{Decode}_B^{O_B}(D_B, x)$ if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$: $Z = Z \cup \{x\}$ output Z	$\xleftarrow{D_B}$	

an interactive 2PC protocol:

Alice (input X, $|X| = n_A$):

for $x \in X$:
  $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
  $S_A = S_A \cup \{(x, pk_x)\}$
$D_A = \mathsf{Encode}_A^{O_A}(S_A)$

$\xrightarrow{D_A}$

Bob (input Y, $|Y| = n_B$):

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{O_B}(S_B)$

$\xleftarrow{D_B}$

for $x \in X$:
  $\widetilde{C}_x = \mathsf{Decode}_B^{O_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

an interactive 2PC protocol:

Alice (input X, $|X| = n_A$):

for $x \in X$:
  $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
  $S_A = S_A \cup \{(x, pk_x)\}$
$D_A = \mathsf{Encode}_A^{O_A}(S_A)$

Bob (input Y, $|Y| = n_B$):

$\xrightarrow{D_A}$

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{O_B}(S_B)$

$\xleftarrow{D_B}$

for $x \in X$:
  $\widetilde{C}_x = \mathsf{Decode}_B^{O_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

"real" interaction:

(corrupt Alice)

Real interaction:
(O_A, O_B simulated honestly)

receive D_A from adversary

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{O_B}(S_B)$

give D_B to adversary

an interactive 2PC protocol:

Alice (input X, $|X| = n_A$):

for $x \in X$:
  $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
  $S_A = S_A \cup \{(x, pk_x)\}$
$D_A = \mathsf{Encode}_A^{O_A}(S_A)$

Bob (input Y, $|Y| = n_B$):

$\xrightarrow{D_A}$

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{O_B}(S_B)$

$\xleftarrow{D_B}$

for $x \in X$:
  $\widetilde{C}_x = \mathsf{Decode}_B^{O_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

"real" interaction:

(corrupt Alice)

Real interaction:
(O_A, O_B simulated honestly)

receive D_A from adversary

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Real interaction:
(O_A, O_B simulated honestly)

receive D_A from adversary

for $y \in Y$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 1:
ExtOracle$^{\mathrm{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $r \leftarrow \mathsf{KEM}.\mathcal{P}_\lambda$
  $Q = Q \cup \{r\}$
  return r

receive D_A from adversary
$\widetilde{X} = \mathrm{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
  $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 2:
ExtOracle$^{\mathrm{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \mathrm{KeyGen}(\lambda)$
  $(\widehat{C}_{pk}, -) = \mathrm{Encaps}(pk)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathrm{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
    $C_y = \widehat{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 3:
ExtOracle$^{\mathrm{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \mathrm{KeyGen}(\lambda)$
  $\widehat{C}_{pk} \leftarrow \mathsf{KEM}.\mathcal{C}_\lambda$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathrm{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
    $C_y = \widehat{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 4:
ExtOracle$^{\mathrm{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \mathrm{KeyGen}(\lambda)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathrm{ExtOKVS}(D_A)$

for $y \in Y \cap \widetilde{X}$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Ideal interaction:
ExtOracle$^{\mathrm{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \mathrm{KeyGen}(\lambda)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathrm{ExtOKVS}(D_A)$

send $\widetilde{X}$ to $\mathcal{F}_{\mathrm{psi}}$
receive $Z = \widetilde{X} \cap Y$ from $\mathcal{F}_{\mathrm{psi}}$

for $y \in Z$:
  $\widetilde{pk}_y = \mathrm{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \mathrm{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathrm{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Real interaction:
(O_A, O_B simulated honestly)

receive D_A from adversary

for $y \in Y$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 1:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $r \leftarrow \text{KEM}.\mathcal{P}_\lambda$
  $Q = Q \cup \{r\}$
  return r

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
  $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 2:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \text{KeyGen}(\lambda)$
  $\widetilde{C}_{pk} \leftarrow \text{KEM}.\mathcal{C}_\lambda$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
    $C_y = \widetilde{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 3:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \text{KeyGen}(\lambda)$
  $\widetilde{C}_{pk} \leftarrow \text{KEM}.\mathcal{C}_\lambda$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin Q$
    $C_y = \widetilde{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Hybrid 4:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \text{KeyGen}(\lambda)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

for $y \in Y \cap \widetilde{X}$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

Ideal interaction:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \text{KeyGen}(\lambda)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

send $\widetilde{X}$ to $\mathcal{F}_{\text{psi}}$
receive $Z = \widetilde{X} \cap Y$ from $\mathcal{F}_{\text{psi}}$

for $y \in Z$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

$\rightsquigarrow$

Ideal interaction:
$\text{ExtOracle}^{\text{Samp}}$ plays role of O_A
(O_B simulated honestly)

Samp():
  $(-, pk) \leftarrow \text{KeyGen}(\lambda)$
  $Q = Q \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \text{ExtOKVS}(D_A)$

send $\widetilde{X}$ to $\mathcal{F}_{\text{psi}}$
receive $Z = \widetilde{X} \cap Y$ from $\mathcal{F}_{\text{psi}}$

for $y \in Z$:
  $\widetilde{pk}_y = \text{Decode}_A^{O_A}(D_A, y)$
  $(C_y, -) = \text{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \text{Encode}_B^{O_B}(S_B)$

give D_B to adversary

open problems

how to display sequences of hybrids on paper?

Uniformly sampled Y-values are indistinguishable from values sampled without replacement. The three-hop maneuver involving lemma 4.5.7 is omitted.

```
PRF.QUERY_H(X):                          PRF.QUERY_H(X):
  if L*[X] undefined:                      if L*[X] undefined:
    Y ←$ {0,1}^λ                             Y ←$ {0,1}^λ \ 𝒴
                                             𝒴 := 𝒴 ∪ {Y}
    if L_0[Y] undefined:                     if L_0[Y] undefined:
      L_0[Y] ←$ {0,1}^λ                        L_0[Y] ←$ {0,1}^λ
    L*[X] := L_0[Y]                          L*[X] := L_0[Y]
  return L*[X]                             return L*[X]
```

In this hybrid, the Y-values are guaranteed to not repeat. Thus, the inner if-statement is *always* taken, and its body can be made unconditional.

```
PRF.QUERY_H(X):                          PRF.QUERY_H(X):
  if L*[X] undefined:                      if L*[X] undefined:
    Y ←$ {0,1}^λ \ 𝒴                         Y ←$ {0,1}^λ \ 𝒴
    𝒴 := 𝒴 ∪ {Y}                             𝒴 := 𝒴 ∪ {Y}
    if L_0[Y] undefined:                   ≡
      L_0[Y] ←$ {0,1}^λ                      L_0[Y] ←$ {0,1}^λ
    L*[X] := L_0[Y]                          L*[X] := L_0[Y]
  return L*[X]                             return L*[X]
```

Now the overall effect of the if-statement's body is to assign a uniformly sampled value to $L^*[X]$. The same logic can be written more directly, without Y, $\mathcal{Y}$, or $L_0[\cdot]$. The result of these simplification is $\mathcal{L}^H_{\text{prf-rand}}$, which completes the proof.

```
                                         𝓛^H_prf-rand
PRF.QUERY_H(X):                          PRF.QUERY_H(X):
  if L*[X] undefined:                      if L*[X] undefined:
    Y ←$ {0,1}^λ \ 𝒴                   ≡
    𝒴 := 𝒴 ∪ {Y}
    L_0[Y] ←$ {0,1}^λ
    L*[X] := L_0[Y]                          L*[X] ←$ {0,1}^λ
  return L*[X]                             return L*[X]
```

Joy of Cryptography

how to display sequences of hybrids on paper?

Uniformly sampled Y-values are indistinguishable from values sampled without replacement. The three-hop maneuver involving lemma 4.5.7 is omitted.

```
PRF.QUERY_H(X):                    ≡   PRF.QUERY_H(X):
  if L*[X] undefined:                    if L*[X] undefined:
    Y ←$ {0,1}^λ                           Y ←$ {0,1}^λ \ 𝒴
                                           𝒴 := 𝒴 ∪ {Y}
    if L_2[Y] undefined:                   if L_2[Y] undefined:
      L_2[Y] ←$ {0,1}^λ                      L_2[Y] ←$ {0,1}^λ
    L*[X] := L_2[Y]                        L*[X] := L_2[Y]
  return L*[X]                           return L*[X]
```

In this hybrid, the Y-values are guaranteed to not repeat. Thus, the inner if-statement is *always* taken, and its body can be made unconditional.

```
PRF.QUERY_H(X):                    ≡   PRF.QUERY_H(X):
  if L*[X] undefined:                    if L*[X] undefined:
    Y ←$ {0,1}^λ \ 𝒴                       Y ←$ {0,1}^λ \ 𝒴
    𝒴 := 𝒴 ∪ {Y}                           𝒴 := 𝒴 ∪ {Y}
    if L_2[Y] undefined:
      L_2[Y] ←$ {0,1}^λ                      L_2[Y] ←$ {0,1}^λ
    L*[X] := L_2[Y]                        L*[X] := L_2[Y]
  return L*[X]                           return L*[X]
```

Now the overall effect of the if-statement's body is to assign a uniformly sampled value to $L^*[X]$. The same logic can be written more directly, without Y, $\mathcal{Y}$, or $L_2[\cdot]$. The result of these simplification is $\mathcal{L}^H_{\text{prf-rand}}$, which completes the proof.

```
                                       𝓛^H_prf-rand
PRF.QUERY_H(X):                    ≡   PRF.QUERY_H(X):
  if L*[X] undefined:                    if L*[X] undefined:
    Y ←$ {0,1}^λ \ 𝒴
    𝒴 := 𝒴 ∪ {Y}
    L_2[Y] ←$ {0,1}^λ
    L*[X] := L_2[Y]
  return L*[X]                           L*[X] ←$ {0,1}^λ
                                         return L*[X]
```

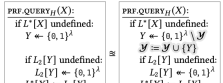
```
KG():                                                    G_2 , G_3
u ← u + 1 ; L_u ←$ {0,1}^k

NAEEnc(i, N_P, N_S, A, M):
C* ‖ T ← AE.Enc(L_i, N_P, N_S, M)
T* ←$ {0,1}^τout \ 𝒯_i
if f_i[N_P ‖ ixor(A, T)] ≠ ⊥ then
  bad ← true ; T* ← f_i[N_P ‖ ixor(A, T)]
𝒯_i ← 𝒯_i ∪ {T*}
f_i[N_P ‖ ixor(A, T)] ← T*
return C* ‖ T*

NAEVer(i, N_P, A, C* ‖ T*):
(M, N_S, T) ← AE.ParDec(L_i, N_P, C*)
if f_i[N_P ‖ ixor(A, T)] ≠ ⊥ then
  if f_i[N_P ‖ ixor(A, T)] = T* then
    bad ← true ; return true
else
  T_d ←$ {0,1}^τout \ 𝒯_i
  if f'_i[N_P ‖ ixor(A, T)] ≠ ⊥ then
    T_d ← f'_i[N_P ‖ ixor(A, T)]
  f'_i[N_P ‖ ixor(A, T)] ← T_d
  𝒯_i ← 𝒯_i ∪ {T_d}
  if T_d = T* then return true
return false
```

Joy of Cryptography

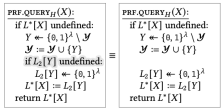
ia.cr/2026/439

how to display sequences of hybrids on paper?

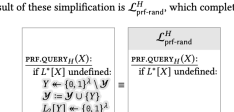
Uniformly sampled Y-values are indistinguishable from values sampled without replacement. The three-hop maneuver involving lemma 4.5.7 is omitted.

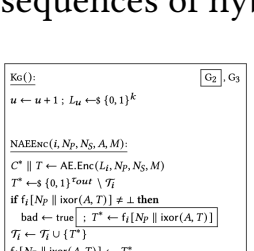


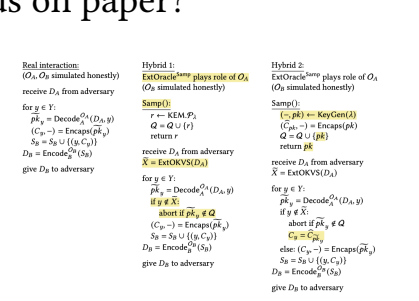
In this hybrid, the Y-values are guaranteed to not repeat. Thus, the inner if-statement is *always* taken, and its body can be made unconditional.



Now the overall effect of the if-statement's body is to assign a uniformly sampled value to $L^*[X]$. The same logic can be written more directly, without Y, $\mathcal{Y}$, or $L_2[\cdot]$. The result of these simplification is $\mathcal{L}^H_{\text{prf-rand}}$, which completes the proof.







Joy of Cryptography ia.cr/2026/439 me + coauthors

adapt to MPC in a more *principled* way

tools for formal proof verification

ProofFrog: A Tool For Verifying Game-Hopping Proofs

Ross Evans
University of Waterloo
Waterloo, Canada
rpevans@uwaterloo.ca

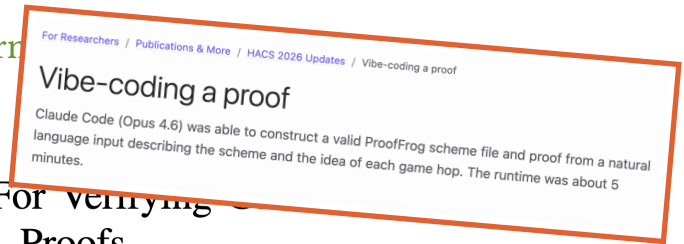
Matthew McKague
Queensland University of Technology
Brisbane, Australia
matthew.mckague@qut.edu.au

Douglas Stebila
University of Waterloo
Waterloo, Canada
dstebila@uwaterloo.ca

ia.cr/2025/418

prooffrog.github.io

tools for form



ProofFrog: A Tool For Verifying Proofs

Ross Evans
University of Waterloo
Waterloo, Canada
rpevans@uwaterloo.ca

Matthew McKague
Queensland University of Technology
Brisbane, Australia
matthew.mckague@qut.edu.au

Douglas Stebila
University of Waterloo
Waterloo, Canada
dstebila@uwaterloo.ca

ia.cr/2025/418
prooffrog.github.io

tools for writing hybrids

$K \twoheadleftarrow \{0,1\}^\lambda$

$\underline{\text{PRF.QUERY}(X_1 \| X_2)}$:	$\underline{\text{PRF.QUERY}(X_1 \| X_2)}$:	$\underline{\text{PRF.QUERY}(X_1 \| X_2)}$:	$\cdots$
	if $L[X_1 \| X_2]$ undef:	if $L[X_1 \| X_2]$ undef:	$\cdots$
		if $R[X_1]$ undef:	$\cdots$
		$R[X_1] \twoheadleftarrow \{0,1\}^n$	$\cdots$
$Y_1 := F(K, X_1)$	$Y_1 := F(K, X_1)$	$Y_1 := R[X_1]$	$\cdots$
		if $R[Y_1 \oplus X_2]$ undef:	$\cdots$
		$R[Y_1 \oplus X_2] \twoheadleftarrow \{0,1\}^n$	$\cdots$
$Y_2 := F(K, Y_1 \oplus X_2)$	$L[X_1 \| X_2] := F(K, Y_1 \oplus X_2)$	$L[X_1 \| X_2] := R[Y_1 \oplus X_2]$	$\cdots$
	return $L[X_1 \| X_2]$	return $L[X_1 \| X_2]$	$\cdots$

edit *vertically* and *horizontally*

1 game-based security

[Shoup04, BellareRogaway06]

1 game-based security

[Shoup04, BellareRogaway06]

write every security definition as

2 indistinguishability of two separate games

1 game-based security

[Shoup04, BellareRogaway06]

write every security definition as

2 indistinguishability of two separate games

3 reductions are implicit

in the game-hopping flow

thanks!