AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

Part 1: Provable Security

1.4: Pseudorandomness

Nadim Kobeissi

https://appliedcryptography.page

Section 1

Pseudorandom Generators

Limitations of One-Time Pad

The Key Length Problem

One-time pad is not a particularly useful encryption scheme in practice.

- The key must be as long as the plaintext!
- This creates a chicken-and-egg situation:
  - To privately send n bits of information,
  - We must already privately share n bits of information.
- Impractical for most real-world applications.
  - Clearly this is not what we're doing when we use HTTPS,
  - or WhatsApp, or pay for something via a debit card...
- We need encryption schemes where the key can be smaller than the message.

Idea: find a way to expand the key

- Given a key k_s of size $|k_s| < |m|$, find a way to obtain $|k_e| \geq |m|$
- In the real world, we have two kinds of symmetric encryption schemes:
  - **Block ciphers**: AES, 3DES, etc.
  - **Stream ciphers**: ChaCha20, RC4, etc.
- This is exactly what stream ciphers do!
  - Start with a small key k_s of a fixed size $|k_s| = \lambda$,
  - Magically expand it to k_e where $|k_e| \geq |m|$,
  - $\textsc{Enc}(K, M) = K \oplus M$

Requirements for Key Expansion

- Our method would need to be **deterministic**, so that both the sender and receiver can expand their key in the same way (to encrypt/decrypt).
- Its output distribution would need to be **uniform**, since that is a crucial property for the security of OTP.
- Unfortunately, it's **not possible** to achieve both of these properties simultaneously.
  - Suppose the expansion method is a deterministic function $G : \{0,1\}^n \to \{0,1\}^{n+\ell}$
  - Its outputs are ℓ bits longer than its inputs!
  - There are $2^{n+\ell}$ strings of length $n+\ell$ but only (at most) 2^n possible outputs of G
  - So the outputs of G can never induce a uniform distribution.

Enter pseudorandomness

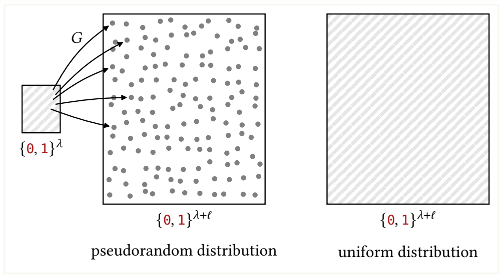
- In cryptography, something is **pseudorandom** if it is indistinguishable from a uniform distribution.[a]
- We need to invent some **"secure pseudorandom generator" (PRG)** G that takes a **seed** S and ends up being indistinguishable from a true uniform distribution when thrown into a library:

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

[a] The Oxford English Dictionary defines the prefix pseudo- as "apparently but not really."

Enter pseudorandomness

- In cryptography, something is **pseudorandom** if it is indistinguishable from a uniform distribution.[a]
- We need to invent some **"secure pseudorandom generator" (PRG)** G that takes a **seed** S and ends up being indistinguishable from a true uniform distribution when thrown into a library:



Source: The Joy of Cryptography

[a] The Oxford English Dictionary defines the prefix pseudo- as "apparently but not really."

Enter pseudorandomness

- We don't know how to make PRGs, or even if they exist.
- So we simply invent functions that we think act close enough to PRGs (when subjected to statistical and mathematical analysis).

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

One-time secrecy of a SKE

One-time Secrecy for SKE

An SKE scheme Σ has one-time secrecy if the following libraries are interchangeable:

$\mathcal{L}^{\Sigma}_{\text{ots-real}}$
$\underline{\text{OTS.ENC}(M)\text{:}}$ $K \twoheadleftarrow \Sigma.\mathcal{K}$ $C := \Sigma.\mathsf{Enc}(K, M)$ return C

$\equiv$

$\mathcal{L}^{\Sigma}_{\text{ots-rand}}$
$\underline{\text{OTS.ENC}(M)\text{:}}$ $C \twoheadleftarrow \Sigma.\mathcal{C}$ return C

An encryption scheme has one-time secrecy if its ciphertexts are uniformly distributed, when keys are sampled uniformly, kept secret, and used for only one encryption, and no matter how the plaintexts are chosen.

Does a PRG-based encryption scheme have one-time secrecy?

If:

$\mathcal{L}_{\text{prg-real}}$
$\text{OTS.ENC}()$:
$S \twoheadleftarrow \{0,1\}^{\lambda}$
return $G(S)$

$\approx$

$\mathcal{L}_{\text{prg-rand}}$
$\text{OTS.ENC}()$:
$Y \twoheadleftarrow \{0,1\}^{\lambda+\ell}$
return Y

Then:

$\mathcal{L}_{\text{ots-real}}$
$\text{OTS.ENC}(M)$:
$K \twoheadleftarrow \{0,1\}^{\lambda}$
$Y := G(K)$
$C := Y \oplus M$
return C

$\approx$

$\mathcal{L}_{\text{ots-rand}}$
$\text{OTS.ENC}(M)$:
$C \twoheadleftarrow \{0,1\}^{\lambda+\ell}$
return C

Attacking PRGs

- Assume that $G(S)$ is a secure PRG.
- Is $\textsc{H}(S)$ secure?

$\underline{\textsc{H}(S)}$:
$A \| B := G(S)$
$C \| D := G(B)$
return $A \| B \| C \| D$

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

$\underline{\textsc{ots.enc}()}$:
$S \twoheadleftarrow \{0,1\}^{\lambda}$
$A \| B := G(S)$
$C \| D := G(B)$
return $A \| B \| C \| D$

$\stackrel{?}{\approxeq}$

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

$\underline{\textsc{ots.enc}(M)}$:
$Y \twoheadleftarrow \{0,1\}^{4\lambda}$
return Y

Attacking PRGs

- Assume that $G(S)$ is a secure PRG.
- Is $\textsc{h}(S)$ secure?
- **No**:
- $\Pr[\mathcal{A} \diamond \mathcal{L}^{H}_{\text{prg-real}} \Rightarrow \texttt{true}] = 1$

$\mathcal{A}$
$A \| B \| C \| D := \textsc{ots.enc}()$
return $G(B) == C \| D$

$\diamond$

$\mathcal{L}^{H}_{\text{prg-real}}$
$\underline{\textsc{ots.enc}():}$
$S \twoheadleftarrow \{0, 1\}^{\lambda}$
$A \| B := G(S)$
$C \| D := G(B)$
return $A \| B \| C \| D$

Attacking PRGs

- Assume that $G(S)$ is a secure PRG.
- Is $\text{H}(S)$ secure?
- **No**:
- $\Pr[\mathcal{A} \diamond \mathcal{L}^{H}_{\text{prg-real}} \Rightarrow \texttt{true}] = 1$
- $\Pr[\mathcal{A} \diamond \mathcal{L}^{H}_{\text{prg-rand}} \Rightarrow \texttt{true}] = \frac{1}{2^{2\lambda}}$
- Difference certainly not negligible.

$\mathcal{A}$
$A\|B\|C\|D := \text{OTS.ENC}()$
return $G(B) == C\|D$

$\diamond$

$\mathcal{L}^{H}_{\text{prg-rand}}$
$\text{OTS.ENC}(M)$:
$Y \twoheadleftarrow \{0,1\}^{4\lambda}$
return Y

Example: RC4 (a stream cipher)

- Used in WEP, SSL/TLS, and other protocols.
- Simple PRG that takes a seed and produces a keystream.
- Basic operation:
  - Initialize S-box with permutation of bytes 0-255.
  - Use key to scramble the S-box.
  - Generate pseudorandom bytes iteratively.
- Several weaknesses found over time:
  - Statistical biases in initial output.
  - Correlation between key and output bytes.
  - Considered cryptographically broken today.

Security Warning

RC4 is presented as a historical example only. It should not be used in new applications due to known weaknesses. Modern alternatives include ChaCha20.

RC4 PRG Algorithm

- RC4 generates a pseudorandom stream of bytes used to encrypt data.
- After key setup (which initializes array S), the algorithm produces keystream bytes:
- Each output byte requires simple operations:
    - Array index calculations.
    - Array value swapping.
    - Modular addition.
- Fast implementation in software.
- Despite simplicity, it has several cryptographic weaknesses.

```python
def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256

        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        yield K
```

RC4 PRG implementation in Python.

Section 2

Pseudorandom Functions

Pseudorandom function: definition

Pseudorandom Function (PRF)

A function $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ is a secure pseudorandom function (PRF) if the following two libraries are indistinguishable:

- n the input length of the PRF.
- m the output length of the PRF.
- λ is the key size and hence the security parameter.

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

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

$\underline{\text{PRF.QUERY}(X)}$:

return $F(K, X)$

$\approx$

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

$L := [\,]$

$\underline{\text{PRF.QUERY}(X)}$:

if $L[X]$ undefined:

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

return $L[X]$

PRG vs PRF: Key Differences

- **PRG (Pseudorandom Generator)**:
  - Takes short seed, produces longer output
  - Generates entire output as a monolithic string
  - $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$

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

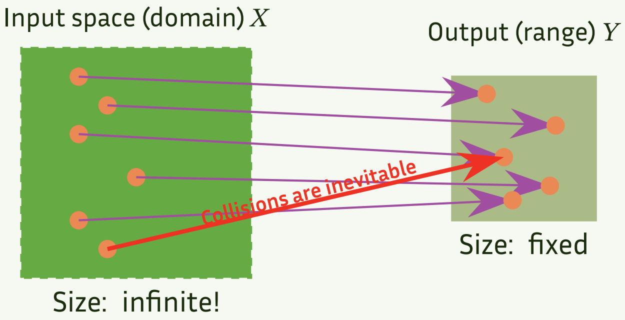
$L := [\,]$

<u>PRF.QUERY(X)</u>:
if $L[X]$ undefined:
  $L[X] \twoheadleftarrow \{0,1\}^m$
return $L[X]$

- **PRF (Pseudorandom Function)**:
  - Maps inputs to pseudorandom outputs
  - Provides access to individual blocks of output
  - Can generate output for any input on demand
  - $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$
- PRFs enable "selective access" to pseudorandom values without generating the entire sequence

PRF : $F_k = X \to Y$

- We want the mapping to be:
  - One-way
  - “Randomized”
  - Relations between inputs not reflected in outputs



PRFs in the real world: hash functions

Hash Function Properties

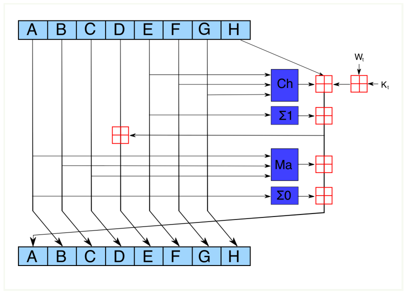
- Takes input of **any size**
- Produces output of **fixed size**
- Is **deterministic** (same input → same output)
- Even a **tiny change** in input creates completely different output
- Is **efficient** to compute

```
SHA256(hello) =
2cf24dba5fb0a30e26e83b2ac5
b9e29e1b161e5c1fa7425e7304
3362938b9824
SHA256(hullo) =
7835066a1457504217688c8f5d
06909c6591e0ca78c254ccf174
50d0d999cab0
```

Note: One character change → completely different hash!

Expected properties of a hash function

- **Collision resistance**: computationally infeasible to find two different inputs producing the same hash.
- **Preimage resistance**: given the output of a hash function, it is computationally infeasible to reconstruct the original input.
- **Second preimage resistance**: given an input and an output, it's computationally infeasible to find another different input producing the same output.



SHA-2 compression function. Source: Wikipedia

Hash functions: what are they good for?

- **Password storage**: Store the hash of the password on the server, not the password itself. Then check candidate passwords against the hash.
- **Data integrity verification**: Hash a file. Later hash it again and compare hashes to check if the file has changed, suffered storage degradation, etc.
- **Proof of work**: Server asks client to hash something a lot of times before they can access some resource. Useful for anti-spam, Bitcoin mining, etc.

An insecure PRF construction

Claim: An Insecure PRF

The function $F(K, X) = G(K) \oplus X$ is not a secure PRF, even if G is a secure PRG.

- This construction fails because:
  - The key K is only fed through the PRG once.
  - The same value $G(K)$ is used for all queries.
  - This creates exploitable patterns in outputs.

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

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

<u>PRF.QUERY(X):</u>
return $G(K) \oplus X$

$\not\approx$

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

$L :=$ [ ]

<u>PRF.QUERY(X):</u>
if $L[X]$ undefined:
  $L[X] \twoheadleftarrow \{0,1\}^{m}$
return $L[X]$

An insecure PRF construction

- When an adversary sees two outputs:

$$Y_1 = G(K) \oplus X_1$$
$$Y_2 = G(K) \oplus X_2$$

- Taking $Y_1 \oplus Y_2$ causes $G(K)$ to cancel:

$$\begin{aligned} Y_1 \oplus Y_2 &= G(K) \oplus X_1 \oplus G(K) \oplus X_2 \\ &= X_1 \oplus X_2 \end{aligned}$$

- In a truly random function, $Y_1 \oplus Y_2 = X_1 \oplus X_2$ would be extremely unlikely!

$\mathcal{L}^F_{\text{prf-real}}$
$K \twoheadleftarrow \{0,1\}^\lambda$
PRF.QUERY(X):
return $G(K) \oplus X$

$\not\approx$

$\mathcal{L}^F_{\text{prf-rand}}$
$L := [\,]$
PRF.QUERY(X):
if $L[X]$ undefined:
$L[X] \twoheadleftarrow \{0,1\}^m$
return $L[X]$

Another insecure PRF construction

Claim: Another Insecure PRF

The function $H(K_1 \| K_2, X_1 \| X_2) = F(K_1, X_1) \oplus F(K_2, X_2)$ is not a secure PRF, even if F is a secure PRF.

- An unsuccessful attempt to use a PRF with shorter input length to build one with a larger input length.

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

$K_1 \| K_2 \twoheadleftarrow \{0,1\}^{2\lambda}$

$\underline{\text{PRF.QUERY}(X_1 \| X_2)}$:

return $F(K_1, X_1) \oplus F(K_2, X_2)$

$\not\approx$

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

$L := [\,]$

$\underline{\text{PRF.QUERY}(X_1 \| X_2)}$:

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

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

return $L[X_1 \| X_2]$

Why the previous PRF is broken

- When inputs share the same first half, the corresponding outputs of H have a common term $F(K_1, X_1)$
- Consider querying four inputs: $A\|B$, $A\|B'$, $A'\|B$, $A'\|B'$ (where $A \neq A'$ and $B \neq B'$)

$$Y_1 = F(K_1, A) \oplus F(K_2, B)$$
$$Y_2 = F(K_1, A) \oplus F(K_2, B')$$
$$Y_3 = F(K_1, A') \oplus F(K_2, B)$$
$$Y_4 = F(K_1, A') \oplus F(K_2, B')$$

- If we XOR $Y_1 \oplus Y_2$, the $F(K_1, A)$ terms cancel out.
- Similarly, $Y_3 \oplus Y_4$ causes $F(K_1, A')$ to cancel:
  - $Y_1 \oplus Y_2 = F(K_2, B) \oplus F(K_2, B')$
  - $Y_3 \oplus Y_4 = F(K_2, B) \oplus F(K_2, B')$
- So $\Pr\left[Y_1 \oplus Y_2 == Y_3 \oplus Y_4\right] = 1$
- With a truly random function, $\Pr\left[Y_1 \oplus Y_2 == Y_3 \oplus Y_4\right] = \frac{1}{2^m}$ (extremely unlikely!)
- Also, Given $Y_{1\ldots3}$, we can predict Y_4!

The Golden Rule of PRFs

The Golden Rule of PRFs

If a PRF F is being used as a component in a larger construction H, then security usually rests on how well H can ensure distinct inputs to F.

- When analyzing PRF security, focus on input uniqueness.
- Repeated inputs to a PRF create exploitable patterns.
- Even if F is secure, H can be broken if it causes F to receive duplicate inputs.
- Don't try to directly distinguish F's outputs from uniform.
- Instead, exploit how H uses F incorrectly.
- Find input patterns that force collisions within F.

Section 3

Pseudorandom Permutations

What is a permutation?

Permutation

A permutation is a rearrangement where each input value maps to exactly one output value, and each possible output appears exactly once.

- Permutations rearrange elements rather than transforming them.
- Every element in the domain appears exactly once in the range.
- The function is invertible.

- **Example**: simple substitution cipher. Each letter maps to another letter:

$$a \mapsto g, \quad b \mapsto a, \quad c \mapsto r$$
$$d \mapsto b, \quad e \mapsto l, \quad \ldots$$

- Under this permutation:
  - "cabbage" $\mapsto$ "rgaagdl"
  - "rgaagdl" $\mapsto$ "cabbage"
- This mapping is reversible because it's a permutation over $\{a, \ldots, z\}$.
- Each letter appears exactly once in the output alphabet.

PRF versus PRP

Pseudo-Random Function (SHA-2)

- **Input** is arbitrary-length,
- **Output** is fixed-length, looks random (as discussed earlier).
- Indistinguishable from a truly random function by an adversary with limited computational power.

Pseudo-Random Permutation (AES)

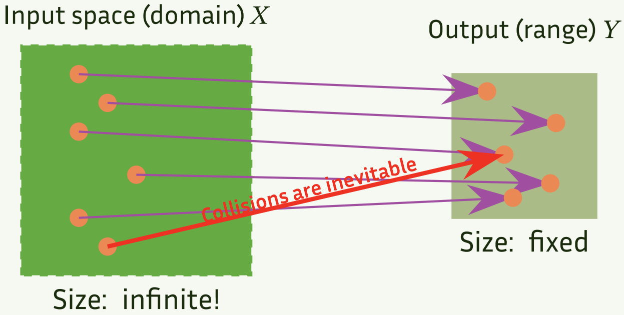
- **Input and output** are the same length, forming a permutation.
- Each input maps uniquely to one output, allowing invertibility.
- Indistinguishable from a truly random permutation by an adversary with limited computational power.

PRPs compared to PRFs

- **Invertibility**: PRPs can be efficiently inverted given the key
  - Enable both encryption and decryption
  - Can recover input from output (and vice versa)
- **No range collision**: Each input maps to a unique output
  - Provides perfect input recovery
  - Reduces vulnerability to collision-based attacks, birthday attacks, and certain forms of differential cryptanalysis
- **Versatility**: A secure PRP can be used as a PRF
  - "Downgrade" trivially by ignoring inverse capability
  - The reverse is not true (PRF → PRP conversion is complex)

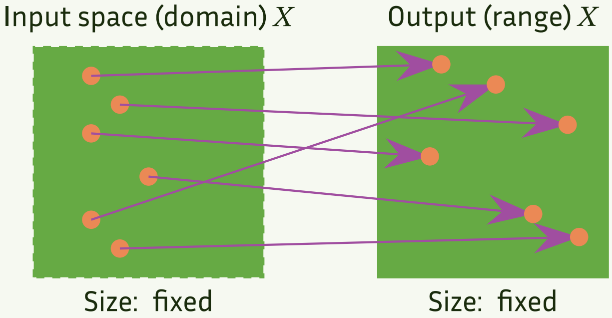
PRF : $F_k = X \to Y$

- We want the mapping to be:
  - One-way
  - “Randomized”
  - Relations between inputs not reflected in outputs



PRP : $F_k = X \to X$

- **Bijective** (two-way)
  - **Injective**: no two inputs map to same output (no collisions)
  - **Surjective**: Every output has one corresponding input
- “Randomized”
- Relations between inputs not reflected in outputs



“Lazy dictionaries” versus “lazy permutations”

Ideal PRF versus ideal PRP

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

$L := [\,]$

PRF.QUERY(X):

if $L[X]$ undefined:

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

return $L[X]$

$\mathcal{L}^{F}_{\text{prp-rand}}$

$L := [\,]$

PRP.QUERY(X):

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^n \setminus y$

  $y := y \cup \{Y\}$

  $L[X] := Y$

return $L[X]$

“Lazy dictionaries” versus “lazy permutations”

- While the PRF (on the left) just picks random outputs for each input...
- The PRP (on the right) must ensure outputs are never repeated:
  - y tracks all outputs used so far
  - $\setminus$ means “set difference” - pick from values not in y
  - $\cup$ means “set union” - add the new value to y
- This ensures each output appears exactly once - the definition of a permutation

$\mathcal{L}^{F}_{\text{prp-rand}}$

$L := [\,]$

$\underline{\text{PRP.QUERY}(X)}$:

if $L[X]$ undefined:

$\quad Y \twoheadleftarrow \{0,1\}^n \setminus y$

$\quad y := y \cup \{Y\}$

$\quad L[X] := Y$

return $L[X]$

Pseudorandom function: definition

Pseudorandom Function (PRF)

A function $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ is a secure pseudorandom function (PRF) if the following two libraries are indistinguishable:

- n the input length of the PRF.
- m the output length of the PRF.
- λ is the key size and hence the security parameter.

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

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

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

$\approx$

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

$L := []$

$\underline{\text{PRF.QUERY}(X)}$:
if $L[X]$ undefined:
  $L[X] \twoheadleftarrow \{0,1\}^m$
return $L[X]$

Pseudorandom permutation: definition

Pseudorandom Permutation (PRP)

A keyed permutation $F^{\pm}$ is a secure pseudo-random permutation (PRP) if the following two libraries are indistinguishable:

- n the input length of the PRP. Also the output length!
- λ is the key size and hence the security parameter.

$\mathcal{L}^{F}_{\text{prp-real}}$

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

PRP.QUERY(X):

return $F(K, X)$

$\approx$

$\mathcal{L}^{F}_{\text{prp-rand}}$

$L := [\,]$

PRP.QUERY(X):

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^{n} \setminus \mathcal{y}$

  $\mathcal{y} := \mathcal{y} \cup \{Y\}$

  $L[X] := Y$

return $L[X]$

Pseudorandom permutation: definition

Pseudorandom Permutation (PRP)

A keyed permutation $F^{\pm}$ is a secure pseudo-random permutation (PRP) if the following two libraries are indistinguishable:

- We obviously can't use $\mathcal{L}^F_{\text{prp-rand}}$ in the real world.
- It doesn't scale. So we need practical approximative alternatives.

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

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

$\underline{\text{PRP.QUERY}(X)}$:

return $F(K, X)$

$\approx$

$\mathcal{L}^F_{\text{prp-rand}}$

$L := [\,]$

$\underline{\text{PRP.QUERY}(X)}$:

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^n \setminus y$

  $y := y \cup \{Y\}$

  $L[X] := Y$

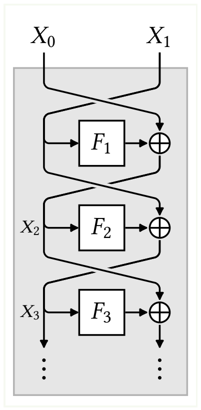
return $L[X]$

Building a permutation through Feistel ciphers

- An r-round **Feistel cipher** with **round functions** $F_1, \ldots, F_r$ is defined as follows:

$$\underline{\mathbb{F}(X_0 \| X_1)}:$$
$$\text{for } i = 1 \text{ to } r:$$
$$\quad X_{i+1} := X_{i-1} \oplus F_i(X_i)$$
$$\text{return } X_r \| X_{r+1}$$

- A Feistel cipher is always a permutation on $\{0, 1\}^{2n}$, regardless of its round functions.



Feistel cipher.
Source: The Joy of Cryptography

Building a permutation through Feistel ciphers

Feistel Ciphers Are Permutations

A Feistel cipher is always a permutation on $\{0, 1\}^{2n}$, regardless of its round functions.

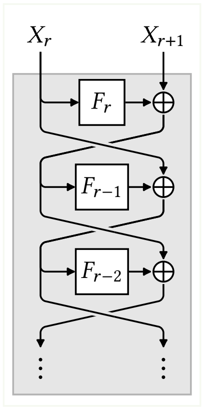
- **Proof:** Each round of a Feistel cipher computes the next block as:
  - $X_{i+1} = X_{i-1} \oplus F_i(X_i)$
- To invert this round, we can rearrange the equation to solve for X_{i-1} in terms of X_i and X_{i+1}:
  - $X_{i-1} = X_{i+1} \oplus F_i(X_i)$
- F_i itself does not need to have an inverse - both the forward and inverse direction of the Feistel cipher evaluate F_i in the forward direction! □

Building a permutation through Feistel ciphers

Inversion

- $\mathbb{F}^{-1}$ is the **inversion**, allowing Feistel ciphers to function as PRPs:

$\underline{\mathbb{F}^{-1}(X_r \| X_{r+1})}$:
for $i = r$ down to 1:
    $X_{i-1} := X_{i+1} \oplus F_i(X_i)$
return $X_0 \| X_1$



Feistel cipher inversion.
Source: The Joy of Cryptography

Forward and backward on one slide

Just in case it's helpful

$\underline{\mathbb{F}(X_0 \| X_1)}$:

for $i = 1$ to r:

$\quad X_{i+1} := X_{i-1} \oplus F_i(X_i)$

return $X_r \| X_{r+1}$

$\underline{\mathbb{F}^{-1}(X_r \| X_{r+1})}$:

for $i = r$ down to 1:

$\quad X_{i-1} := X_{i+1} \oplus F_i(X_i)$

return $X_0 \| X_1$

Keyed Feistel ciphers

- Let's add a key in there. Wow, encryption!
- $K_{1\ldots i}$ called is the **key schedule**.
- If each round function F_i uses a distinct key K_i, it increases the security of the Feistel network against certain attacks.
- By using a PRF as round function F_i, the security of the Feistel cipher would be grounded on the PRF's security basis.

$\underline{\mathbb{F}(K_1 \| \cdots \| K_r,\ X_0 \| X_1)}$:

for $i = 1$ to r:

$\quad X_{i+1} := X_{i-1} \oplus F_i(K_i, X_i)$

return $X_r \| X_{r+1}$

Meet-in-the-middle attacks on Feistel ciphers

Why use a key schedule and not the same key?

- Using the **same key** for all rounds creates significant vulnerabilities.
- A meet-in-the-middle attack can break an r-round Feistel cipher with complexity $> \frac{2^{|K|}}{2}$.
- The attack works by computing partial encryptions from both ends:
  1. Forward: Compute halfway through encryption.
  2. Backward: Compute halfway through decryption.
  3. Look for "meeting points" in the middle.
- With identical round functions, effective security may be only **half** the number of rounds.

$\underline{\mathbb{F}(K,\ X_0 \| X_1)}$:

for $i = 1$ to r:

$\quad X_{i+1} := X_{i-1} \oplus F(K, X_i)$

return $X_r \| X_{r+1}$

Breaking a 2-round Feistel cipher

- **A 2-round Feistel cipher cannot be a secure PRP.**

$\underline{\mathbb{F}(K_1 \| K_2,\ X_0 \| X_1)}$:

$X_2 := X_0 \oplus F(K_1, X_1)$

$X_3 := X_1 \oplus F(K_2, X_2)$

return $X_2 \| X_3$

Breaking a 2-round Feistel cipher

- Once again, real or random?
- We can indeed produce an adversary $\mathcal{A}$ that can distinguish between these two libraries.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \twoheadleftarrow \{0,1\}^{2\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:

$X_2 := X_0 \oplus F(K_1, X_1)$

$X_3 := X_1 \oplus F(K_2, X_2)$

return $X_2 \| X_3$

$\not\approx$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-rand}}$

$L := [\,]$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X)}$:

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^n \setminus y$

  $y := y \cup \{Y\}$

  $L[X] := Y$

return $L[X]$

Breaking a 2-round Feistel cipher

- Let's query $(A \| B)$ and $(A' \| B)$ where $A \neq A'$
  - $Y_1 \| Y_2 = \text{PRP.QUERY}(A \| B)$
  - $Y_1' \| Y_2' = \text{PRP.QUERY}(A' \| B)$
- In a 2-round Feistel cipher:
  - $Y_1 = A \oplus F(K_1, B)$
  - $Y_1' = A' \oplus F(K_1, B)$
  - $Y_1 \oplus Y_1' = A \oplus A'$
- In a true PRP, $\Pr[Y_1 \oplus Y_1' == A \oplus A']$ is negligible

$\mathcal{A}$
$Y_1 \| Y_2 = \text{PRP.QUERY}(A \| B)$ $Y_1' \| Y_2' = \text{PRP.QUERY}(A' \| B)$ return $Y_1 \oplus Y_1' == A \oplus A'$

$\diamond$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$
$K_1 \| K_2 \twoheadleftarrow \{0, 1\}^{2\lambda}$ $\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$: $X_2 := X_0 \oplus F(K_1, X_1)$ $X_3 := X_1 \oplus F(K_2, X_2)$ return $X_2 \| X_3$

However, a 3+ round Feistel cipher is fine!

$\mathcal{A}$

$Y_1 \| Y_2 = \text{PRP.QUERY}(A \| B)$
$Y_1' \| Y_2' = \text{PRP.QUERY}(A' \| B)$
return $Y_1 \oplus Y_1' == A \oplus A'$

$\diamond$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \twoheadleftarrow \{0,1\}^{2\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:
$X_2 := X_0 \oplus F(K_1, X_1)$
$X_3 := X_1 \oplus F(K_2, X_2)$
return $X_2 \| X_3$

$\not\approx$

$\mathcal{A}$

$Y_1 \| Y_2 = \text{PRP.QUERY}(A \| B)$
$Y_1' \| Y_2' = \text{PRP.QUERY}(A' \| B)$
return $Y_1 \oplus Y_1' == A \oplus A'$

$\diamond$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \| K_3 \twoheadleftarrow \{0,1\}^{3\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:
$X_2 := X_0 \oplus F(K_1, X_1)$
$X_3 := X_1 \oplus F(K_2, X_2)$
$X_4 := X_2 \oplus F(K_3, X_3)$
return $X_3 \| X_4$

However, a 3+ round Feistel cipher is fine!

- Luby and Rackoff proved that a 3-round Feistel cipher is indistinguishable from a pseudorandom permutation.[a]
- Can we also prove it using our provable security framework?

[a] https://appliedcryptography.page/papers/luby-rackoff.pdf

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \| K_3 \twoheadleftarrow \{0,1\}^{3\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:

$X_2 := X_0 \oplus F(K_1, X_1)$

$X_3 := X_1 \oplus F(K_2, X_2)$

$X_4 := X_2 \oplus F(K_3, X_3)$

return $X_3 \| X_4$

$\approx$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-rand}}$

$L := [\,]$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X)}$:

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^n \setminus y$

  $y := y \cup \{Y\}$

  $L[X] := Y$

return $L[X]$

The “bad event” proof technique

Reminder

Bad Event Technique

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be libraries that each include a boolean variable named bad, and assume that after bad is set to `true` it remains `true` forever. We say that the bad event is triggered if the library ever sets `bad := true`.

If $\mathcal{L}_1$ and $\mathcal{L}_2$ have identical source code, except for statements reachable only when `bad := true`, then:

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow \texttt{true}] - \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow \texttt{true}]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_1 \text{TRIGGER}(\texttt{bad})]$$

The “bad event” proof technique

Reminder

- $\mathcal{A}$'s advantage is bounded by $\Pr[\mathcal{A} \diamond \mathcal{L}_1\text{TRIGGER}(\mathsf{bad})]$.
- Practical application:
  - Define a sequence of hybrid libraries.
  - Identify “bad events” between consecutive hybrids.
  - Show these events occur with negligible probability.
- Enables us to focus on analyzing specific failure cases rather than full behavior.

The "end-of-time" strategy for bad events

Reminder

- Sometimes analyzing bad events can be complex, especially when values are chosen by the adversary.
- The end-of-time strategy:
    1. Postpone all bad-event logic to the end of the library execution.
    2. Collect information during normal execution.
    3. Check for bad events only at the very end.
- Advantages:
    - Simplifies analysis by separating normal behavior from bad-event checking.
    - Makes it easier to bound the probability of bad events.
    - Particularly useful for complex cryptographic proofs.

However, a 3+ round Feistel cipher is fine!

- Luby and Rackoff proved that a 3-round Feistel cipher is indistinguishable from a pseudorandom permutation.[a]
- Can we also prove it using our provable security framework?
- Yes, with the bad events proof technique!

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \| K_3 \twoheadleftarrow \{0,1\}^{3\lambda}$

$\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)$:

$X_2 := X_0 \oplus F(K_1, X_1)$

$X_3 := X_1 \oplus F(K_2, X_2)$

$X_4 := X_2 \oplus F(K_3, X_3)$

return $X_3 \| X_4$

$\approx$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-rand}}$

$L := [\,]$

$\text{PRP.QUERY}_{\mathbb{F}}(X)$:

if $L[X]$ undefined:

  $Y \twoheadleftarrow \{0,1\}^n \setminus y$

  $y := y \cup \{Y\}$

  $L[X] := Y$

return $L[X]$

[a] https://appliedcryptography.page/papers/luby-rackoff.pdf

Proof of security for 3-round Feistel cipher

Step 1

- We start at $\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \| K_3 \twoheadleftarrow \{0, 1\}^{3\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:

$X_2 := X_0 \oplus F(K_1, X_1)$

$X_3 := X_1 \oplus F(K_2, X_2)$

$X_4 := X_2 \oplus F(K_3, X_3)$

return $X_3 \| X_4$

Proof of security for 3-round Feistel cipher

Step 2

- We add a cache $L[\cdot]$ so that each distinct output is computed only once.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$K_1 \| K_2 \| K_3 \twoheadleftarrow \{0,1\}^{3\lambda}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:

if $L[X_0 \| X_1]$ undefined:

$\quad X_2 := X_0 \oplus F(K_1, X_1)$

$\quad X_3 := X_1 \oplus F(K_2, X_2)$

$\quad X_4 := X_2 \oplus F(K_3, X_3)$

$\quad L[X_0 \| X_1] := X_3 \| X_4$

return $L[X_0 \| X_1]$

Proof of security for 3-round Feistel cipher

Step 3

- We add additional sub-cache $L_i[\cdot]$ for each $F(K_i, \cdot)$
- Since we already assume F to be a secure PRF, we can replace it with the ideal PRF and remove K.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)$:
if $L[X_0 \| X_1]$ undefined:
    if $L_1[X_1]$ undefined:
        $L_1[X_1] \twoheadleftarrow \{0,1\}^\lambda$
    $X_2 := X_0 \oplus L_1[X_1]$
    if $L_2[X_2]$ undefined:
        $L_2[X_2] \twoheadleftarrow \{0,1\}^\lambda$
    $X_3 := X_1 \oplus L_2[X_2]$
    if $L_3[X_3]$ undefined:
        $L_3[X_3] \twoheadleftarrow \{0,1\}^\lambda$
    $X_4 := X_2 \oplus L_3[X_3]$
    $L[X_0 \| X_1] := X_3 \| X_4$
return $L[X_0 \| X_1]$

Proof of security for 3-round Feistel cipher

Step 4

- We expect X_2 and X_3 to never repeat. So, we can trigger the bad event if they do.
- Later, we must show that the bad event's probability is negligible.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:
if $L[X_0 \| X_1]$ undefined:
  if $L_1[X_1]$ undefined:
    $L_1[X_1] \twoheadleftarrow \{0,1\}^\lambda$
  $X_2 := X_0 \oplus L_1[X_1]$
  if $L_2[X_2]$ defined: bad := true
  $L_2[X_2] \twoheadleftarrow \{0,1\}^\lambda$
  $X_3 := X_1 \oplus L_2[X_2]$
  if $L_3[X_3]$ defined: bad := true
  $L_3[X_3] \twoheadleftarrow \{0,1\}^\lambda$
  $X_4 := X_2 \oplus L_3[X_3]$
  $L[X_0 \| X_1] := X_3 \| X_4$
return $L[X_0 \| X_1]$

Proof of security for 3-round Feistel cipher

Step 5

- Instead of sampling $L_2[X_2]$ uniformly and then computing X_3, we can sample X_3 uniformly and compute $L_2[X_2]$.
- Same for $L_3[X_3]$ and X_4.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

```
PRP.QUERY_F(X0‖X1):
if L[X0‖X1] undefined:
    if L1[X1] undefined:
        L1[X1] ← {0,1}^λ
    X2 := X0 ⊕ L1[X1]
    if L2[X2] defined: bad := true
    X3 ← {0,1}^λ
    L2[X2] := X3 ⊕ X1
    if L3[X3] defined: bad := true
    X4 ← {0,1}^λ
    L3[X3] := X4 ⊕ X2
    L[X0‖X1] := X3‖X4
return L[X0‖X1]
```

Proof of security for 3-round Feistel cipher

Step 6

- We can move the sampling steps to the top since they're no longer dependent on other variables.
- Note how nothing after $L[X_0 \| X_1] := X_3 \| X_4$ affects what the adversary sees!

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

$\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)$:
if $L[X_0 \| X_1]$ undefined:
  $X_3 \twoheadleftarrow \{0,1\}^\lambda$
  $X_4 \twoheadleftarrow \{0,1\}^\lambda$
  $L[X_0 \| X_1] := X_3 \| X_4$
  if $L_1[X_1]$ undefined:
    $L_1[X_1] \twoheadleftarrow \{0,1\}^\lambda$
  $X_2 := X_0 \oplus L_1[X_1]$
  if $L_2[X_2]$ defined: bad := true
  $L_2[X_2] := X_3 \oplus X_1$
  if $L_3[X_3]$ defined: bad := true
  $L_3[X_3] := X_4 \oplus X_2$
return $L[X_0 \| X_1]$

Proof of security for 3-round Feistel cipher

Step 7

- We can move the sampling steps to the top since they're no longer dependent on other variables.
- Note how nothing after $L[X_0 \| X_1] := X_3 \| X_4$ affects what the adversary sees!
- So, we can move all bad-event logic to the end of time, without changing the bad event's overall probability.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

```
PRP.QUERY_F(X0‖X1):
if L[X0‖X1] undefined:
    L[X0‖X1] := {0,1}^2λ
    𝒳 := 𝒳 ∪ {X0‖X1}
    return L[X0‖X1]

END OF TIME():
for each X0‖X1 ∈ 𝒳 :
    X3‖X4 := L[X0‖X1]
    if L1[X1] undefined:
        L1[X1] ← {0,1}^λ
    X2 := X0 ⊕ L1[X1]
    if L2[X2] defined: bad := true
    L2[X2] := X3 ⊕ X1
    if L3[X3] defined: bad := true
    L3[X3] := X4 ⊕ X2
```

Proof of security for 3-round Feistel cipher

Step 8

- We need to analyze the probability of the bad event happening.
- Let's say the adversary makes q queries.
- The bad event happens if:
  - X_2 value collides with previous X_2.
  - X_3 value collides with previous X_3.
- Recall: $X_2 = X_0 \oplus L_1[X_1]$ where $L_1[X_1]$ is chosen randomly.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

PRP.QUERY$_{\mathbb{F}}(X_0 \| X_1)$:
if $L[X_0 \| X_1]$ undefined:
  $L[X_0 \| X_1] := \{0,1\}^{2\lambda}$
  $\mathcal{X} := \mathcal{X} \cup \{X_0 \| X_1\}$
  return $L[X_0 \| X_1]$

END OF TIME():
for each $X_0 \| X_1 \in \mathcal{X}$:
  $X_3 \| X_4 := L[X_0 \| X_1]$
  if $L_1[X_1]$ undefined:
    $L_1[X_1] \twoheadleftarrow \{0,1\}^{\lambda}$
  $X_2 := X_0 \oplus L_1[X_1]$
  if $L_2[X_2]$ defined: bad := true
  $L_2[X_2] := X_3 \oplus X_1$
  if $L_3[X_3]$ defined: bad := true
  $L_3[X_3] := X_4 \oplus X_2$

Proof of security for 3-round Feistel cipher

Step 9

- So, we need to analyze when X_2 collisions occur.
- If (X_0, X_1) and (X_0', X_1') are two different inputs...
- A collision happens when $X_2 = X_2'$:

$$X_0 \oplus L_1[X_1] = X_0' \oplus L_1[X_1']$$

- If $X_1 \neq X_1'$, then $L_1[X_1]$ and $L_1[X_1']$ are independent random values
- The probability of this specific collision is $2^{-\lambda}$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

```
PRP.QUERY_F(X0‖X1):
if L[X0‖X1] undefined:
    L[X0‖X1] := {0,1}^2λ
    𝒳 := 𝒳 ∪ {X0‖X1}
    return L[X0‖X1]

END OF TIME():
for each X0‖X1 ∈ 𝒳 :
    X3‖X4 := L[X0‖X1]
    if L1[X1] undefined:
        L1[X1] ↞ {0,1}^λ
    X2 := X0 ⊕ L1[X1]
    if L2[X2] defined: bad := true
    L2[X2] := X3 ⊕ X1
    if L3[X3] defined: bad := true
    L3[X3] := X4 ⊕ X2
```

Proof of security for 3-round Feistel cipher

Step 10 - Total probability analysis

- For q queries, we have at most $q(q-1)/2$ pairs of queries
- Prob. of any X_2 collision across q queries: $q(q-1)/(2 \cdot 2^\lambda) \approx q^2/2^{\lambda+1}$
- Similarly for X_3 values: the probability of any X_3 collision is at most $q^2/2^{\lambda+1}$.
- Total probability of bad event: at most $q^2/2^\lambda$
- With $\lambda \gg \log q$, this probability is negligible.

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

```
PRP.QUERY_F(X_0||X_1):
if L[X_0||X_1] undefined:
    L[X_0||X_1] := {0,1}^{2λ}
    𝒳 := 𝒳 ∪ {X_0||X_1}
    return L[X_0||X_1]

END OF TIME():
for each X_0||X_1 ∈ 𝒳 :
    X_3||X_4 := L[X_0||X_1]
    if L_1[X_1] undefined:
        L_1[X_1] ← {0,1}^λ
    X_2 := X_0 ⊕ L_1[X_1]
    if L_2[X_2] defined: bad := true
    L_2[X_2] := X_3 ⊕ X_1
    if L_3[X_3] defined: bad := true
    L_3[X_3] := X_4 ⊕ X_2
```

Proof of security for 3-round Feistel cipher

Step 11

- By the bad event technique, the advantage of any adversary is at most $q^2/2^\lambda$.
- Without bad events, our last hybrid samples each response randomly.
- Since the sampling logic is all that remains visible to the adversary, this is equivalent to the final simplified library.
- This is indistinguishable from a truly random permutation for $q \ll 2^{\lambda/2}$ queries.
- (The birthday bound tells us that's when collisions become likely)

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$

PRP.QUERY$_{\mathbb{F}}(X_0 \| X_1)$:

```
if L[X_0||X_1] undefined:
    L[X_0||X_1] := {0,1}^{2λ}
    𝒳 := 𝒳 ∪ {X_0||X_1}
    return L[X_0||X_1]
```

END OF TIME():

```
for each X_0||X_1 ∈ 𝒳 :
    X_3||X_4 := L[X_0||X_1]
    if L_1[X_1] undefined:
        L_1[X_1] ← {0,1}^λ
    X_2 := X_0 ⊕ L_1[X_1]
    if L_2[X_2] defined: bad := true
    L_2[X_2] := X_3 ⊕ X_1
    if L_3[X_3] defined: bad := true
    L_3[X_3] := X_4 ⊕ X_2
```

Proof of security for 3-round Feistel cipher

- Rest of the transition steps are trivial.

  □

$\mathcal{L}^{\mathbb{F}}_{\text{prp-real}}$
$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X_0 \| X_1)}$:
if $L[X_0 \| X_1]$ undefined:
$\quad L[X_0 \| X_1] := \{0,1\}^{2\lambda}$
$\quad$ return $L[X_0 \| X_1]$

$\approx$

$\mathcal{L}^{\mathbb{F}}_{\text{prp-rand}}$
$L := [\,]$
$\underline{\text{PRP.QUERY}_{\mathbb{F}}(X)}$:
if $L[X]$ undefined:
$\quad Y \twoheadleftarrow \{0,1\}^n \setminus y$
$\quad y := y \cup \{Y\}$
$\quad L[X] := Y$
return $L[X]$

DES: Feistel in practice

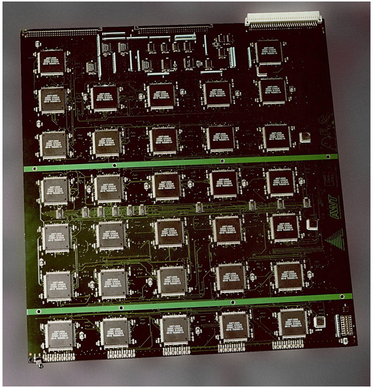
- **Data Encryption Standard (DES)** is a classic real-world implementation of the Feistel structure
- Properties:
  - 16 Feistel rounds.
  - 56-bit key (64 bits with parity).
  - 64-bit block size.
  - Standardized in 1977.
- Problem: By the 1990s, 56-bit keys became too small for security.

- **Triple DES (3DES)** replaced it:
  - Uses three DES operations in sequence.
  - $C = E_{K3}(D_{K2}(E_{K1}(P)))$
  - Effectively doubles the key length.
  - Compatible with legacy DES when $K1 = K2 = K3$
- 3DES still used in legacy systems, but largely replaced by AES for performance reasons.
- Lesson: Feistel structure made it possible to adapt DES rather than abandon it.

EFF's "Deep Crack"

- In 1998, the Electronic Frontier Foundation built a special-purpose machine called "Deep Crack".[a]
- Cost: Only $250,000 (far less than the NSA budget!)
- Purpose: Prove that 56-bit DES keys were insufficient.

[a] I'm not responsible for any readings into that name.



EFF's "Deep Crack"

EFF's "Deep Crack"

- July 1998: Deep Crack broke a DES challenge in just 56 hours.
- Message revealed: "It's time for those 128-, 192-, and 256-bit keys."
- Impact:
  - Publicly demonstrated DES was obsolete.
  - Accelerated adoption of AES.
  - Made "it's theoretically breakable" a practical reality.
  - Priceless reaction from government officials!

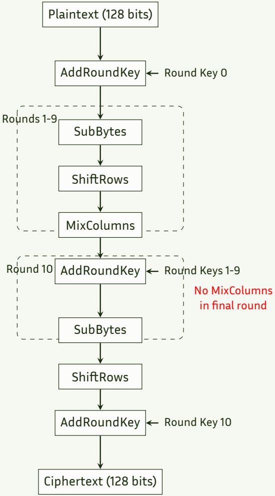
Paul Kocher. He's still active in cryptography today, insanely productive research career, many crazy attacks

AES: a good example of a PRP

- AES is the most widely used PRP in the world.
- It works on fixed-size blocks: 128 bits.
- Key sizes: 128, 192, or 256 bits.
- Each AES key defines a specific permutation over the space of all 128-bit values.
- For each key, AES maps each possible 128-bit input to exactly one 128-bit output.
- Different keys create different permutations.
- AES is efficiently invertible:
  - $\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$
- AES is believed to be computationally indistinguishable from a random permutation.
- Has withstood extensive cryptanalysis for over 20 years.

AES structure

- Internal structure: substitution-permutation network with multiple rounds.[a]
  - SubBytes: non-linear substitution
  - ShiftRows: transposition
  - MixColumns: mixing operation
  - AddRoundKey: XOR with round key



[a] Check out this amazing interactive animation of AES's internal structure: `https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html`

AES: security and attacks over time

- AES has been heavily analyzed for over 20 years.
- Best attacks against full AES have gradually improved:
  - 2011: Biclique attack (Bogdanov et al.) reduced complexity to $2^{126.1}$ for AES-128.
  - Various side-channel attacks developed (power analysis, cache timing).[a]
  - Advances in meet-in-the-middle and related-key techniques.

[a] This is the main way to attack AES in practice. Side-channel attacks will be discussed in more depth later in the course.

- Despite these advances:
  - No practical attacks on full AES-128.
  - Best attacks still require $\approx 2^{126}$ operations.
  - At this complexity, attacks remain purely theoretical.
  - Would require resources far exceeding global computing power.
- Even quantum computers offer only modest advantage (Grover's algorithm reduces security to 2^{64} operations).[a]

[a] More on quantum computers and how they affect cryptography later in the course.

AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

Part 1: Provable Security

1.4: Pseudorandomness

Nadim Kobeissi

https://appliedcryptography.page