CS 6160 Cryptology Lecture 7: Practical Constructions of Block Ciphers

Maria Francis

September 28, 2021

Block Ciphers

- A block cipher must behave like a random permutation.
- On ℓ-bit strings, there $2^\ell!$ permutations possible and to represent a permutation we need $\ell \cdot 2^\ell$ bits.
- For $\ell > 20$ it is impractical and infeasible for $\ell > 50$. For modern block ciphers block length ℓ is ≥ 128.
- We need a set of permutations with a concise description (a short key) but still behaves like a random permutation.
- Changing one bit in the input of E_k should yield an independent result, should affect every bit of output.
- Does not mean every output bit should be changed! That is not random. Every bit is changed with probability roughly half.

Confusion-Diffusion paradigm

- Shannon again! He gave a way for constructing concise random-looking permutations.
- We need to construct a random looking permutation E with a large block length from many random looking (or random) permutations $\{f_i\}$ with small block length.
- For e.g: We need to build E with block length 128 bits.
- The key k for E will specify 16 permutations $f_1, \ldots, f_{16}$ that have 8-bit block length.
- Given $x \in \{0,1\}^{128}$, we parse it as 16-bytes $x_1 \ldots x_{16}$ and then define,

$$E_K(x) = f_1(x_1) \circ \cdots \circ f_{16}(x_{16}).$$

$\{f_i\}$s introduce *confusion* into F.

Confusion-Diffusion paradigm

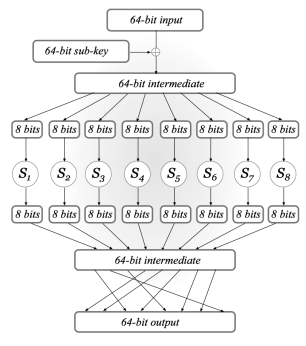
- Note that permutation on 8 bits can be represented using $\log(2^8!) \approx 1600$ bits much smaller than $128 \cdot 2^{128}$ bits.
- But E is not pseudorandom. For eg: if x and $x^{'}$ differ only in their first bit then $E_k(x)$ and $E_k(x^{'})$ will differ only in their first byte *regardless of the key*.
- This is why we need *diffusion*, output bits are permuted using a mixing permutation to spread a local change everywhere in the block.
- The confusion diffusion steps constitute a round and such rounds are repeated several times.

Substitution-Permutation Networks (SPN)

- SPN implements the confusion-diffusion paradigm.
- For round functions we give it a particular form rather than choosing all possible permutations on say 8-bit strings.
- We fix a substitution function S called S-box and using k we define $S(k \oplus x)$.
- For example: consider a SPN with a 64-bit block length based on a collection of 8-bit S-boxes $S_1, \ldots, S_8$.
- Each round we do the following operations on 64-bit input m:
  - Key mixing: Set $x = m \oplus k$, where k is the current-round sub-key.
  - Substitution: Set $x = S_1(x_1) \circ \cdots \circ S_8(x_8)$ where x_i is the ith byte of x.
  - Permutation: Permute all the bits of x to obtain the output of the round which is the input of next round.

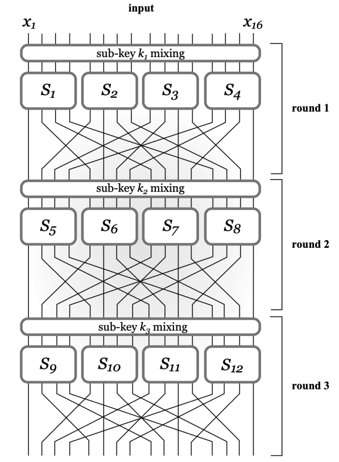
SPN rounds

- After the final round, there is also a key-mixing step and then we get the final output of the cipher.
- Kerckhoff's principle: S-boxes and mixing permutations are public.
- The key-mixing step is key, *the substitution-permutation steps add no security as no key is involved*.



SPN multiple rounds

High level structure of an SPN with 16-bit block length and 4-bit S-boxes in each round.



SPN

- The actual key of the block cipher is called master key.
- For each round a round key is generated. They derived from the master key using a key schedule.
- It can be a simple algorithm : take different subsets of the bits of the master key. Or could be a complex relation.
- An r-round SPN has r full rounds of key-mixing, S-box substitution, application of a mixing permutation and teh final key-mixing step.
- We need then $r+1$ sub-keys.

SPN

- SPN is invertible given the key, i.e. given the output of a SPN and the key we can retrieve the input.
- Note that if a single round is inverted then the entire SPN can be inverted.
- How to invert a single round?
  - invert the mixing permutation, which is just rearrangement of bits
  - S-boxes are permutations too and therefore one-one, which means it can be inverted too.
  - The result XOR with sub-key gives original input of the round.

Thus we have:
Let E be a keyed function defined by an SPN in which S-boxes are permutations. Then regardless of the key schedule and the number of rounds E_k is a permutation for any k.

Security of SPN – Avalanche Effect

- The number of rounds, design of S-boxes, mixing permutations, key scheduling algo are important in determining the security.
- Avalanche effect for block ciphers : a small change in the input must "affect" every bit of the output.
- How to implement it in SPN ?
  - S-boxes are designed so that changing a single bit of its input changes at least two bits of the S-box.
  - Mixing permutations are such that the output bits of any given S-box are used as input to multiple S-boxes in the next round.

How does it yield the avalanche effect?

- Assume the S-boxes have 8 bits input and output and block length of the cipher is 128 bits.
- Consider what happens when you are giving two inputs that differ in a single bit.
- After the first round, the intermediate values differ in exactly two bit positions. How?
    - XORing with sub-key maintains 1-bit difference, so the input to S-box is identical except for one.
    - For that S-box since there is 1-bit difference, the output will generate a 2-bit difference.
    - The mixing permutation applied to the results rearranges positions but maintains the 2-bit difference.

How does it yield the avalanche effect?

- After the first round, the mixing permutation spreads the two bit-positions to two different S-boxes in the second round.
- This is the case even after XORing with the sub-key for the next round.
- Now in the second round two S-boxes receive inputs differing in single input, thus resulting in an intermediate value that differ in 4 bits.
- Same argument means 8 bits after 3 rounds, 16 bits after 4th round and all 128 bits after 7th round.
- The arguments are not that precise and there could be fewer differences than expected at the end of some round, so we need to use more than 7 rounds.
- But 7 gives us a lower bound: less than that and we can distinguish this cipher from a random permutation.

How to design S-boxes?

- Choose them at random? Not the best way!
  - Let a S-box S be s.t. it takes 4-bit inputs and let x and $x^{'}$ be two distinct values with $y = S(x)$
  - Consider choosing uniform $y^{'}(\neq y)$ as $y^{'} = S(x^{'})$.
  - There are 4 strings that differ from y in only 1 bit and so $\frac{4}{15}$ is the probability that we will choose $y^{'}$ that does not differ from y in more than 1 bit.
  - When we consider all pairs that differ in exactly one bit, the probability is compounded.
- So S-boxes have to be carefully designed!
- For a block cipher to be strongly pseudorandom, the avalanche effect should apply to its inverse as well.
  Changing every bit of the *output* should affect every bit of the input. Answer: More rounds!

Feistel Networks

- Another approach for constructing block ciphers.
- Major advantage/difference over SPN: the underlying functions (analogous to S-boxes) need not be invertible.
- It gives a way to construct an invertible function from non-invertible components.
- This helps give the block cipher an "unstructured" behavior as requiring all the components to be invertible introduces structure.
- Also invertibility makes designing S-boxes hard.

Feistel Networks

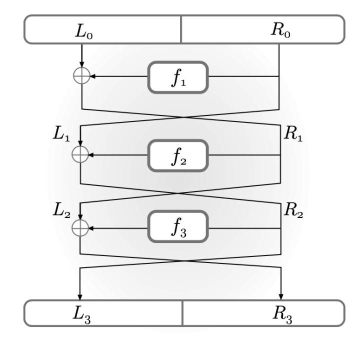
- It operates with a series of rounds.
- In each round, a keyed round function is applied.
- They need not be invertible.
- Typically build from S-boxes and mixing permutations but a Feistel network can deal with any round function.
- Balanced Feistel networks –
  - ith round - $\hat{f}_i$ takes sub-key k_i and $\ell/2$ bit string and outputs an $\ell/2$-bit string.
  - Master key k is used to derive sub-keys k_i for each round i.

$$f_i : \{0,1\}^{\ell/2} \to \{0,1\}^{\ell/2}$$

$$f_i(x) = \hat{f}_i(k_i, x)$$

ith round of Feistel Networks

- The input ℓ-bit is divided into two $\ell/2$ bit halves: left: L_{i-1} and right: R_{i-1}
- Output (L_i, R_i) is $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f_i(R_{i-1})$.
- For r-round Feistel Network, the ℓ-bit input is parsed as (L_0, R_0) and the output is ℓ-bit (L_r, R_r).



Inverting a Feistel Network

- It is invertible regardless of the $\{f_i\}$ (and thus $\hat{f}_i$)
- To show that a Feistel network is invertible it is enough to show that each round of the network is invertible given $\{f_i\}$.
- Given the output (L_i, R_i) of the ith round, we show how to compute (L_{i-1}, R_{i-1}):

$$R_{i-1} := L_i$$
$$L_{i-1} := R_i \oplus f_i(R_{i-1})$$

Note that we took f_i, not its inverse.

Let E be a keyed function defined by a Feistel network. Then regardless of the round functions $\{\hat{f}_i\}$ and the number of rounds, E_k is an efficiently invertible permutation for all k.

DES : Data Encryption Standard

- The most common example for a block cipher, but an old one (1970s) and is considered a legacy encryption scheme.
- Evolved from IBM's submission, Lucifer, to the NIST's request for encryption algorithms that could be standardized.
- A well-engineered algorithm with an immense influence in the field.
- Short $k = 56$ bits and $n = 64$ bits and has 16 rounds of a Feistel network.
- Triple DES (3DES) introduced in 1998 was used in finance and payment industries until July 2017 when NIST proposed retiring it. In Nov 2017, NIST allowed for restricted usage, no longer be used in TLS, IPsec or for large files.

DES

- General consensus: except for short key everything else is well-designed.
- Best attack in practice is exhaustive search over 2^{56} keys.
- There are theoretical attacks that we will discuss that assumes less computation but these attacks assume certain conditions that are difficult to realize in practice.
- We give a high-level overview of the design to understand the basic ideas but it is not a full specification!

Design of DES

- DES cipher has a 16 round Feistel cipher, block length $\ell = 64$ bits and key length $n = 56$ bits.
- The same round function $\hat{f}$ is used in each round.
- Each round function takes a 48-bit sub-key and a 32-bit input (half a block) since it is a balanced Feistel network.
- Key scheduling algorithm, *KeySchedule* takes the 56-bit key and gives as output 48 bit length sub-keys, $k_1, \ldots, k_{16}$.
- *KeySchedule* algorithm : gives k_i a permuted subset of 48 bits of the master key.
  - Divide 56 bit master key to left and right halves of 28 bits.
  - For the leftmost 24 bits of k_i, we take some subset of the 28 bits of the left half of master key.
  - And the same for the rightmost 24 bits of k_i.
  - How it is done is all public!

Key Scheduling of DES

The permutations applied on the master key:

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

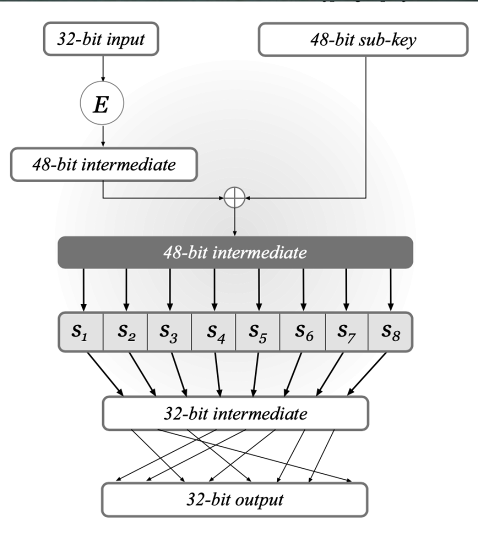
The *KeySchedule* algorithm:

Algorithm $KeySchedule(K)$ // $|K| = 56$
    $K \leftarrow PC\text{-}1(K)$
    Parse K as $C_0 \| D_0$
    for $r = 1, \ldots, 16$ **do**
        if $r \in \{1, 2, 9, 16\}$ **then** $j \leftarrow 1$ else $j \leftarrow 2$ **fi**
        $C_r \leftarrow leftshift_j(C_{r-1})$; $D_r \leftarrow leftshift_j(D_{r-1})$
        $K_r \leftarrow PC\text{-}2(C_r \| D_r)$
    return$(K_1, \ldots, K_{16})$

DES Round Function

- Also called DES mangler function : a substitution-permutation network.
- Each round the input to $\hat{f}$ is $k_i \in \{0,1\}^{48}$ and $x \in \{0,1\}^{32}$.
- We first expand x to a 48-bit value $x^{'}(= Exp(x))$, by simply duplicating half the bits of x.
- We then do computation just like before in SPN:
  - $x^{'}$ is XORed with k_i which is 48 bits long.
  - Divide the result into 8 blocks each of 6 bits long.
  - Each 6-bit block is passed into a S-box and yields a 4-bit output.
  - Concatenate the result of 8 S-boxes and we have a 32-bit result.
  - Then a mixing permutation.
- Main difference: S-boxes are not invertible, inputs are longer than outputs.

DES Round Function



S-boxes in DES

- The eight S-boxes are the non-linear part of DES and they were very carefully designed. Even a small change makes DES vulnerable to attacks.
- Each S-box takes a 6-bit input and gives a 4-bit output.
- View it as a table with 4 rows and 16 columns with each entry a 4-bit entry.
- The $4 \cdot 16 = 64 = 2^6$ entries of the table can be indexed by the 6-bit input: 1 and 6 bits for row and the other bits for column.

S-boxes in DES

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_1$:	0	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	1	0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	1	1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_2$:	0	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	0	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	1	0	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	1	1	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_3$:	0	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	0	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	1	0	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	1	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_4$:	0	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	0	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	1	0	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	1	1	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_5$:	0	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	0	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	1	0	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	1	1	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_6$:	0	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	0	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	1	0	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	1	1	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_7$:	0	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	0	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	0	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	1	1	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathbf{S}_8$:	0	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	0	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	1	0	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	1	1	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES avalanche effect

- Changing one input bit to a S-box changes at least two output bits.
- The mixing permutation ensures that the four output bits of any S-box will affect the input of 6 S-boxes of next round!
- Consider two inputs (L_0, R_0) and (L_0', R_0) which differ by only a single bit in the left half.
- After first round, the intermediate values only differ by one bit but now the difference is in the right half.
- If you assume that the bit is not duplicated in the expansion step, the input to S-box differs only in one bit again.
- This generatesa two bit difference after S-box computation.
- This means after two rounds there is a total of 3 bits difference.

DES avalanche effect

- The mixing permutation spreads the two bit difference of right halves to two different S-boxes, resulting in a 4-bit difference in the right halves. And of course you have a 2-bit difference in left halves.
- As with a SPN, after 7 rounds all 32 bits in the right half and after 8 rounds all 32 bits in the left half are affected.
- With 16 rounds we are good!
- How to attack reduced rounds of both SPN and DES?
- Note that DES is a Feistel construction but has only dependent weak pseudorandom round functions.
- Luby-Rackoff Theorem: *Three rounds of Feistel construction with independent PRFs as round functions makes the block cipher a weak PRP and four rounds makes it a strong PRP.*

How to improve DES?

- Ultimately every attack for DES is exhaustive search of key space!
- Increase key size to 128 and then use a different key schedule but still choosing a 48-bit sub-key!
- Or change the S-boxes and use a larger sub-key for each round.
- Any small seemingly insignificant change and we are worried about its security!
- Preferred approach: use DES as a blackbox and try and build something that invokes this DES.

Double Encryption

- Let E be a block cipher with an n-bit key and ℓ-bit block length.
- We define a new cipher, $E^{'}$ with key length $2n$:

$$E^{'}_{k_1,k_2}(x) := E_{k_2}(E_{k_1}(x)),$$

  where k_1, k_2 are independent keys.
- For DES, we get a 2DES which takes a 112 bit key.
- Looks good since time 2^{112} is out of reach.
- Meet-in-the-middle Attack : an attack that needs time roughly 2^n.

Meet-in-the-middle Attack

Assume the adversary is given a single input/output pair (x, y),
$y = E'_{k_1^*,k_2^*}(x) = E_{k_2^*}(E_{k_1^*}(x))$, for unknown k_1^*, k_2^*.

- For each $k_1 \in \{0,1\}^n$, compute $z := E_{k_1}(x)$ and store (z, k_1) in a list L.
- For each $k_2 \in \{0,1\}^n$, compute $z := E_{k_2}^{-1}(y)$ and store (z, k_2) in a list L'.
- We say $(z_1, k_1) \in L$ and $(z_2, k_2) \in L'$ are a match if $z_1 = z_2$. For each match add (k_1, k_2) to a set S.
- This attacks takes time $\mathcal{O}(n \cdot 2^n)$ and requires space $\mathcal{O}((n + \ell) \cdot 2^n)$.
- Also (k_1^*, k_2^*) is in S.
- With more input/output pairs we can identify the key with high probability.

Triple Encryption

What if I apply the block cipher three times?

- Variant 1: Choose three independent keys k_1, k_2, k_3,

$$E''_{k_1,k_2,k_3} = E_{k_3}(E^{-1}_{k_2}(E_{k_1}(x))).$$

- Variant 2: Choose three independent keys k_1, k_2, k_3,

$$E''_{k_1,k_2} = E_{k_1}(E^{-1}_{k_2}(E_{k_1}(x))).$$

Note that the second E is inverse permutation. This is not an issue since if E is a strong pseudorandom function then E^{-1} is also one.

Security of the Variants

- Variant 1: Key length is $3n$, but security is only 2^{2n} because we can do meet-in-the-middle attack.
- Not as secure as $3n$ keys but still secure for practical purposes.
- Variant 2:Key length is $2n$ and it gives security against attacks running in time 2^{2n}.
- Why use inverse? For backward compatibility. Set $k_1 = k_2 = k_3$ then we get a single invocation of E with k_1.
- Triple DES (3DES) : Standardized in 1999 and widely used even now.
- It can be any of the two variants.
- It has relatively small block length and it is relatively slow.

AES – Advanced Encryption Standard

- In Jan 1997, NIST announced a competition for a new block cipher – AES.
- In Oct 2000, Rijndael algorithm designed by two Belgian cryptographers Rijmen and Daemen was the final choice and which later after some modifications became AES.
- Winner was selected based on efficiency, performance in hardware, flexibility, etc.
- AES block cipher has a 128-bit block length and can be used with 128, 192, 256-bit keys.
- The length of key affects the key schedule, the number of rounds but not the high-level.

AES – Advanced Encryption Standard

- AES uses SPN not Feistel network.
- During computation a 4-by-4 array of bytes called state is modified.
- The first state is defined by the input which is 128 bits or 16 bytes.
    1. AddRoundKey: Every round a 128-bit subkey is derived from the master key and is interpreted as a 4-by-4 array. XOR it with the sub-key to get new state array.
    2. SubBytes: Each byte of the state array is replaced by another byte according to a S-box, a bijection over $\{0,1\}^8$.
    3. ShiftRows: Bytes in each row of the state array are shifted to the left.
    4. MixColumns: An invertible transformation (linear transformation over a finite field, $GF(2^8)$) is applied to the four bytes in each column. Each input byte affects all four output bytes. Together with ShiftRows, it provides diffusion.

AES – Advanced Encryption Standard

- MixColumns: If two inputs differ in b bytes then the transformation yields outputs differing in at least $5 - b$ bytes.
- In the final round, MixColumns is replaced with AddRoundKey: to avoid the adversary from inverting last three stages which are not dependent on the key.
- Essentially AES is a SPN :
  - the input is XORed with the round key
  - a small, invertible function is applied to chunks of the resulting value
  - And then mix the bits of result to obtain diffusion.

Security of AES

- Number of rounds : 10 for 128 bit key, 12 for 192 bit key, 14 rounds for 256 bit key.
- To date, there are no practical cryptanalytic attacks that are significantly better than an exhaustive search for the key.
- But no simple hardness assumption for the security for AES.
- AES-256 is widely believed to be secure against quantum computers too!
  - ▶ In a quantum computer, the time taken for a search problem scales slower than for a classical one: search for an n-bit key is equiv to a $n/2$-bit key. So AES-256 effectively has a 128-bit key against a quantum attacker, so still ok!
- AES-128, AES-192, AES-256 : in IPSec, TLS, iOS Keychain
- 3DES: some Mac OSes for their Keychain

AES and DES as PRPs

- DES:

$$K \times X \longrightarrow X, X = \{0,1\}^{64}, K = \{0,1\}^{56}$$

- AES-128/192/256:

$$K \times X \longrightarrow X, K = \{0,1\}^{128/192/256}, X = \{0,1\}^{128}$$

- These PRFs are not proven to be secure PRFs.
- They have been subjected to intense scrutiny by cryptographers so we assume they are secure PRFs.