Theoretical construction of symmetric key primitives

Presentation by:
V. Balasubramanian
SSN College of Engineering

Objectives

- One way functions
- From One-Way Functions to Pseudo randomness
- Constructing Pseudorandom Generators

One way functions

A one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is easy to compute, yet hard to invert. The first condition is easy to formalize: we will simply require that f be computable in polynomial time. Since we are ultimately interested in building cryptographic schemes that are hard for a probabilistic polynomial-time adversary to break except with negligible probability, we will formalize

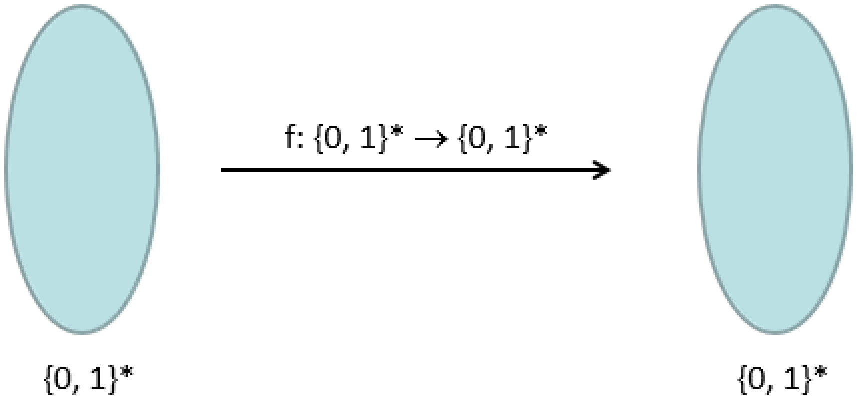
OWF

$$f(x) = y$$

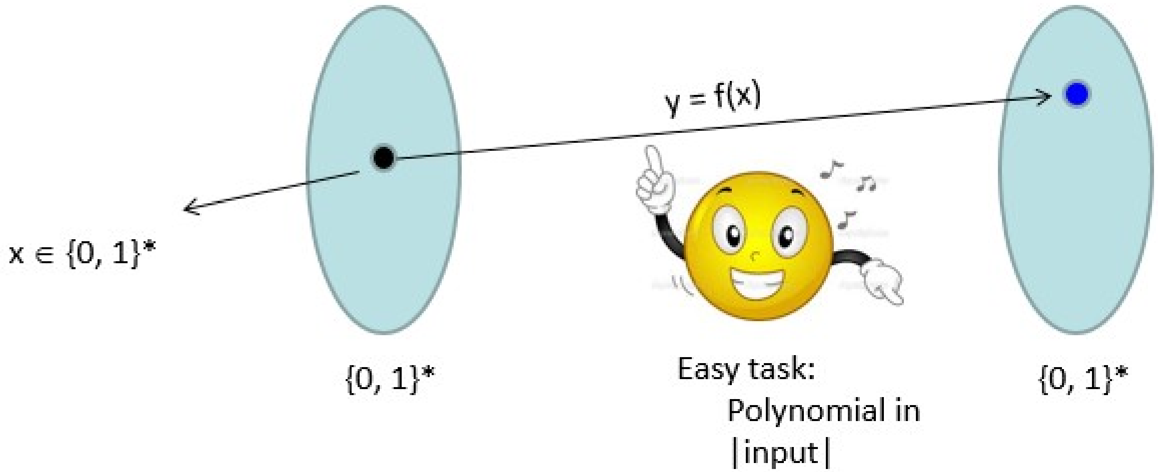
Definition: A function f: $\{0,1\}^* \rightarrow \{0,1\}^*$ is one way if it is

1. **(Easy to compute)** There is a polynomial time algorithm (in |x|) for computing f(x).
2. **(Hard to Invert)** Select $x \leftarrow \{0,1\}^n$ uniformly at random and give the attacker input 1^n, f(x). The probability that a PPT attacker outputs x' such that $f(x') = f(x)$ is negligible.

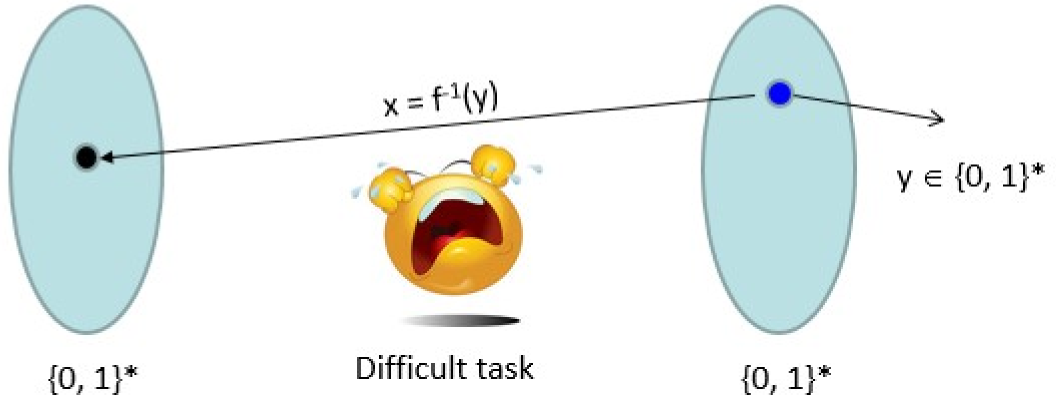
Functions that are easy to compute but “difficult” to invert (almost-always)



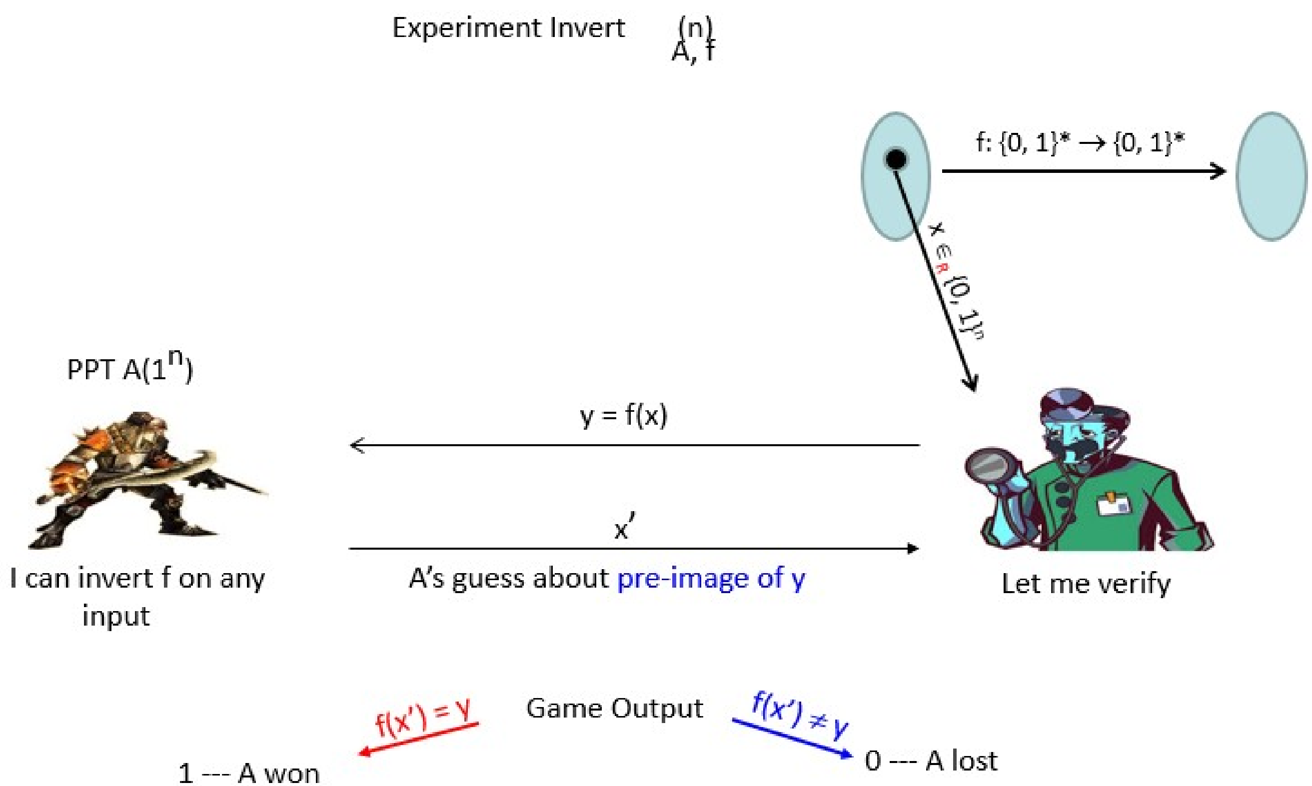
OWF



OWF



Experiment



ıeed not have to find the original x to win the game --- sufficient to find one pre-image

OWF Mathematical function

$$\{0, 1\}^* \xrightarrow{f: \{0, 1\}^* \to \{0, 1\}^*} \{0, 1\}^*$$

Function f is a OWF if the following two conditions hold :

- Easy to compute: for every $x \in \{0, 1\}^*$, $f(x)$ can be computed in poly($|x|$) times

- Hard to Invert: For every PPT algorithm A, there is a negligible function negl() :

$$\Pr\left(\mathrm{Invert}_{A,f}(n) = 1\right) \leq \mathrm{negl}(n) \quad \approx \quad \Pr_{x \leftarrow \{0, 1\}^n}\left[A(f(x), 1^n) \in f^{-1}(f(x))\right] \leq \mathrm{negl}(n)$$

- OWF does not exist in the realm of unbounded powerful adversary.
  - Any function is invertible in principle given enough time/computational power.
  - The assumption of existence of OWF is about computational hardness.

Not -OWF

$$\Pr\left(\mathrm{Invert}_{A,f}(n) = 1\right) \leq \mathrm{negl}(n) \quad \approx \quad \Pr_{x \leftarrow \{0,1\}^n}[\,A(f(x), 1^n) \in f^{-1}(f(x))\,] \leq \mathrm{negl}(n)$$

For a function to be non-OWF, there should exist an A, p(n) s.t

$$\Pr_{x \leftarrow \{0,1\}^n}[\,A(f(x), 1^n) \in f^{-1}(f(x))\,] \geq 1/p(n) \text{ for infinite many } n\text{'s}$$

Example Integer Factorization

Example II: $f(x, y) = x.\ y$, where $x, y \in N$

f is not one-way

$\Pr_{x, y \leftarrow \{0,1\}^{n/2}} [\ A(f(x, y), 1^n) \in f^{-1}(f(x, y))] \geq 3/4$

xy: even $\rightarrow$ (2, xy/2) is a pre-image)

means, in turn, that there exists a positive polynomial $p(\cdot)$ such that for *infinitely many values of* n, algorithm $\mathcal{A}$ inverts f with probability at least $1/p(n)$. Thus, if there exists an $\mathcal{A}$ that inverts f with probability n^{-10} for all even values of n (but always fails to invert f when n is odd), then f is not one-way—even though $\mathcal{A}$ only succeeds on half the values of n, and only succeeds with probability n^{-10} (for values of n where it succeeds at all).

Do OWF Exist?

No unconditional proof of their existence yet.

- Proof is hard because existence of OWF $\rightarrow$ P $\neq$ NP

- Finding a proof will lead to solving the million dollar question in CS whether P = NP or not

Whole world believes that they do and so existence of OWF is an assumption/conjecture

- Several noteworthy computational problems (int. factorization) received intensive attention since ages (even before crypto was born) but no poly time algo is found.

- P $\neq$ NP $\nrightarrow$ existence of OWF

  > The former suggests every PPT algo must fail to solve at least for one input

  > The latter suggests every PPT algo must fail to solve ALMOST ALWAYS (for any random input)

- Being NP-complete is not enough to be a candidate OWF

- Belief that OWF exists is much more than believing P $\neq$ NP

- Non-existence of OWF $\nrightarrow$ P = NP But, P = NP $\rightarrow$ non-existence of OWF

Do OWF Exists?

- OWF $\rightarrow$ P $\neq$ NP
- P $\neq$ NP $\nrightarrow$ OWF
- no OWF $\nrightarrow$ P = NP
- P = NP $\rightarrow$ no OWF

Definition

Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function. Consider the following experiment defined for any algorithm $\mathcal{A}$ and any value n for the security parameter:

The inverting experiment $\mathsf{Invert}_{\mathcal{A},f}(n)$

1. *Choose uniform* $x \in \{0,1\}^n$, *and compute* $y := f(x)$.
2. $\mathcal{A}$ *is given* 1^n *and* y *as input, and outputs* x'.
3. *The output of the experiment is defined to be* 1 *if* $f(x') = y$, *and* 0 *otherwise.*

We stress that $\mathcal{A}$ need not find the original preimage x; it suffices for $\mathcal{A}$ to find any value x' for which $f(x') = y = f(x)$. We give the security parameter

Definition

DEFINITION 7.1 *A function* $f : \{0,1\}^* \rightarrow \{0,1\}^*$ *is* one-way *if the following two conditions hold:*

1. **(Easy to compute:)** *There exists a polynomial-time algorithm* M_f *computing* f*; that is,* $M_f(x) = f(x)$ *for all* x.
2. **(Hard to invert:)** *For every probabilistic polynomial-time algorithm* $\mathcal{A}$*, there is a negligible function* negl *such that*

$$\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n).$$

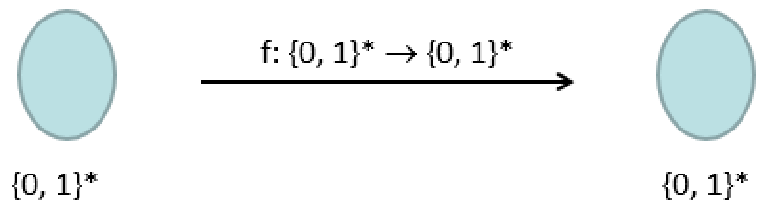
Exponential Time

Exponential-time inversion. Any one-way function can be inverted at any point y in exponential time, by simply trying all values $x \in \{0,1\}^n$ until a value x is found such that $f(x) = y$. Thus, the existence of one-way functions is inherently an assumption about *computational complexity* and *computational hardness*. That is, it concerns a problem that can be solved in principle but is assumed to be hard to solve efficiently.

Additional Property OW Permutation

One-way permutations. We will often be interested in one-way functions with additional structural properties. We say a function f is *length-preserving* if $|f(x)| = |x|$ for all x. A one-way function that is length-preserving and one-to-one is called a *one-way permutation*. If f is a one-way permutation, then any value y has a unique preimage $x = f^{-1}(y)$. Nevertheless, it is still hard to find x in polynomial time.

OWP



Function f is length-preserving if $|f(x)| = |x|$ for all x

- Size of the image and pre-image are the same

Function f is a OWP if it is a OWF and

- Length-preserving
- One-to-one mapping

If f is a OWP then every y has a unique pre-image x

- Still finding x should be hard in polynomial time

OWP

$$f(x) = y$$

Remarks:

- **A function that is not one-way is not necessarily always easy to invert (even often)**
- **Any such function can be inverted in time 2^n (brute force)**
- **Length-preserving OWF: $|f(x)| = |x|$**
- **One way permutation: Length-preserving + one-to-one**

Hash function

DEFINITION 7.2 *A tuple* $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ *of probabilistic polynomial-time algorithms is a* function family *if the following hold:*

1. *The* parameter-generation algorithm Gen, *on input* 1^n, *outputs parameters* I *with* $|I| \geq n$. *Each value of* I *output by* Gen *defines sets* $\mathcal{D}_I$ *and* $\mathcal{R}_I$ *that constitute the domain and range, respectively, of a function* f_I.

2. *The* sampling algorithm Samp, *on input* I, *outputs a uniformly distributed element of* $\mathcal{D}_I$.

3. *The deterministic* evaluation algorithm f, *on input* I *and* $x \in \mathcal{D}_I$, *outputs an element* $y \in \mathcal{R}_I$. *We write this as* $y := f_I(x)$.

Adversary

Let Π be a function family. What follows is the natural analogue of the experiment introduced previously.

The inverting experiment $\mathsf{Invert}_{\mathcal{A},\Pi}(n)$**:**

1. *$\mathsf{Gen}(1^n)$ is run to obtain I, and then $\mathsf{Samp}(I)$ is run to obtain a uniform $x \in \mathcal{D}_I$. Finally, $y := f_I(x)$ is computed.*
2. *$\mathcal{A}$ is given I and y as input, and outputs x'.*
3. *The output of the experiment is 1 if $f_I(x') = y$.*

Example

solving them. Perhaps the most famous such problem is *integer factorization*, i.e., finding the prime factors of a large integer. It is easy to multiply two numbers and obtain their product, but difficult to take a number and find its factors. This leads us to define the function $f_{\mathrm{mult}}(x, y) = x \cdot y$. If we do not place any restriction on the lengths of x and y, then f_{mult} is easy to invert: with high probability $x \cdot y$ will be *even*, in which case $(2, xy/2)$ is an inverse. This issue can be addressed by restricting the domain of f_{mult} to equal-length *primes* x and y. We return to this idea in Section 8.2.

Conclusion

Finally, we remark that very efficient one-way functions can be obtained from practical cryptographic constructions such as SHA-1 or AES under the assumption that they are collision resistant or a pseudorandom permutation,

Function

DEFINITION 7.4 *A function* hc : $\{0,1\}^* \rightarrow \{0,1\}$ *is a* hard-core predicate of a function f *if* hc *can be computed in polynomial time, and for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \mathsf{hc}(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the uniform choice of x in $\{0,1\}^n$ and the randomness of $\mathcal{A}$.

Hard Core Predicates

By definition, a one-way function is hard to invert. Stated differently: given $y = f(x)$, the value x cannot be computed *in its entirety* by any polynomial-time algorithm (except with negligible probability; we ignore this here). One might get the impression that nothing about x can be determined from $f(x)$ in polynomial time. This is *not* necessarily the case. Indeed, it is possible for $f(x)$ to "leak" a lot of information about x even if f is one-way. For a trivial

Hard Core Predicates

- Let f: $\{0, 1\}^* \to \{0, 1\}^*$ be a permutation

- Let hc: $\{0, 1\}^* \to \{0, 1\}$ be a Boolean function

Function hc is a hard-core predicate for the permutation f if the following holds:

- Given x, the value hc(x) can be computed in polynomial (in input size) time

- $\Pr_{x \leftarrow \{0, 1\}^n} [\, A(f(x), 1^n) = hc(x)\,] \leq \frac{1}{2} + negl(n)$

Hard Core

$$f(x) = y$$

Remarks:

1. **f(x) does not necessarily hide all information about x.**
2. **If f(x) is one way then so is** $\mathbf{f}'(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \parallel \boldsymbol{LSB}(\boldsymbol{x})$.

OWF Candidate

This issue can be addressed by restricting the domain of f_{mult} to equal-length *primes* x and y. We return to this idea in Section 8.2.

f(x, y) = xy : x and y are equal length primes.

$$f_{p,g}(x) = [g^x \mod p]$$

(Discrete Logarithm Problem)

Corollary: If one-way functions exist then PRGs, PRFs and strong PRPs all exist.

Corollary: If one-way functions exist then there exist CCA-secure encryption schemes and secure MACs.

OWF to Pseudorandomness

Theorem: Given a one-way-permutation f and a hard-core predicate hc we can construct a PRG G with expansion factor $\ell(n) = n + 1$.

Construction:

$$G(s) = f(s) \parallel \mathrm{hc}(s)$$

Intuition: f(s) is actually uniformly distributed

- s is random
- f(s) is a permutation
- Last bit is hard to predict given f(s) (since hc is hard-core for f)

Theorem: Suppose that there is a PRG G with expansion factor $\ell(n) = n + 1$. Then for any polynomial p(.) there is a PRG with expansion factor p(n).

Theorem: Suppose that there is a PRG G with expansion factor $\ell(n) = 2n$. Then there is a secure PRF.

Theorem: Suppose that there is a secure PRF then there is a strong pseudorandom permutation.

PRG,PRF,PRP

When you think about PRF (Pseudo Random Function), you will think that there are three elements with PRF, which are K, X, and Y. K is the keyspace, X the message or input space and Y the output space. PRF is a function, when you give this function elements from K and X, it will output an element from Y:

$$F : K \times X \to Y$$

When you think about PRP (Pseudo Random Permutation), it also has three elements with PRP, which are K, X, X. As you see the input *and* output space are X:

$$E : K \times X \to X$$

Also, a PRP is required to be bijective, and to have an efficient inversion function PRP^{-1}. This makes sense when recalling that PRPs are sometimes called a *blockcipher*. The inversion function is (needed to build) the decryption function of a blockcipher.

PRFs and PRPs are both deterministic: Calling a PRF or a PRP again a same input as before will produce the same output, respectively.

The inversion function is an important difference between PRF and PRP.

OWF-PRF

PRF

PRG: G: $\{0,1\}^n \rightarrow \{0,1\}^{poly(n)}$

PRG G: $\{0,1\}^n \rightarrow \{0,1\}^{n+1}$

OWF/P g, hc

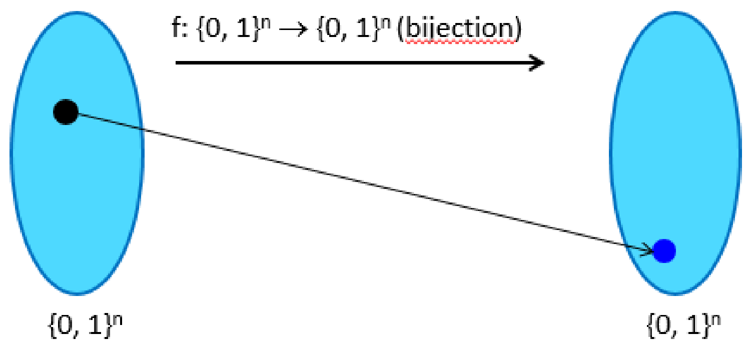
OWF/P f

From One-Way Functions to Pseudo randomness

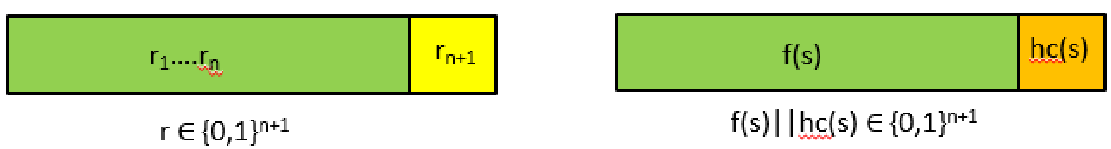
THEOREM 7.7 *If there exists a pseudorandom generator with expansion factor $\ell(n) = n + 1$, then for any polynomial* poly *there exists a pseudorandom generator with expansion factor* $\mathsf{poly}(n)$.

Constructing Pseudorandom Generators

Theorem: Let f be a OWP with hard-core predicate hc. Then the algorithm $G(s) = f(s)||hc(s)$ is a PRG with expansion factor n+1



- s uniform random $\to$ f(s) uniformly random
- Given f(s), the value hc(s) is close to random



- First n bits have same dist. (purely random)
- Last bit is random in r but "close to" random in the latter

- Theorem: Let f be a OWP with hard-core predicate hc. Then the algorithm $G(s) = f(s)||hc(s)$ is a PRG with expansion factor $l(n) = n+1$

PRF

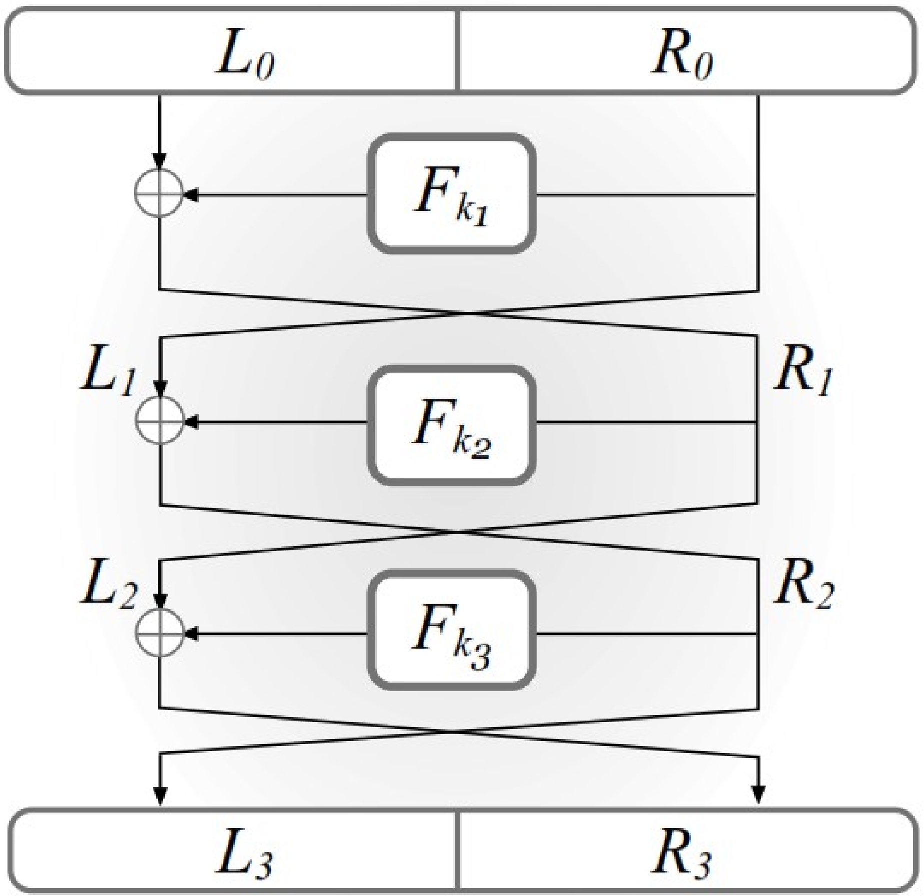
CONSTRUCTION 7.21

Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$, and define G_0, G_1 as in the text. For $k \in \{0,1\}^n$, define the function $F_k : \{0,1\}^n \to \{0,1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}\left(\cdots (G_{x_2}(G_{x_1}(k)))\cdots\right).$$

A pseudorandom function from a pseudorandom generator.

Fiestel Networks



THEOREM 7.25 *If F is a pseudorandom function, then Construction 7.24 is a strong pseudorandom permutation that maps $2n$-bit inputs to $2n$-bit outputs (and uses a $4n$-bit key).*

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(4)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3, k_4)$ with $|k_i| = n$, and an input $x \in \{0,1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**

  1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
  2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
  3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
  4. Compute $L_4 := R_3$ and $R_4 := L_3 \oplus F_{k_4}(R_3)$.
  5. Output (L_4, R_4).

Assumptions

THEOREM 7.26 *If one-way functions exist, then so do pseudorandom generators, pseudorandom functions, and strong pseudorandom permutations.*

THEOREM 7.27 *If one-way functions exist, then so do CCA-secure private-key encryption schemes and secure message authentication codes.*

Pseudorandomness implies one-way functions. We begin by showing that pseudorandom generators imply the existence of one-way functions:

PROPOSITION 7.28 *If a pseudorandom generator exists, then so does a one-way function.*

ssn