Function Families

- A function family is a map

\[ F: \text{Keys}(F) \times \text{Dom}(F) \rightarrow \text{Range}(F) \]

where

- \text{Keys}(F) is the set of keys of \( F \), and is finite
- \text{Dom}(F) is the domain of \( F \)
- \text{Range}(F) is the range of \( F \), and is finite

- For any \( K \in \text{Keys}(F) \), we define \( F_K: \text{Dom}(F) \rightarrow \text{Range}(F) \) by \( F_K(X) = F(K, X) \)

- Usually
  - \text{Keys}(F) = \{0,1\}^k \text{, where } k \in \mathbb{N} \text{ is the key length}
  - \text{Dom}(F) = \{0,1\}^l \text{, where } l \in \mathbb{N} \text{ is the input length}
  - \text{Range}(F) = \{0,1\}^L \text{, where } L \in \mathbb{N} \text{ is the output length}
Function Families

- There is some probability distribution on $\text{Keys}(F)$
  - We denote by $K \leftarrow \text{Keys}(F)$ the selection of a key according to this distribution and its assignment to $K$
  - We denote by $f \leftarrow F$ the operation: $K \leftarrow \text{Keys}(F) ; f \leftarrow F_K$
  - Usually, this distribution is the uniform distribution

- $F$ is a family of permutations if
  - $\text{Dom}(F) = \text{Range}(F)$
  - Each $F_K$ is a permutation, i.e., $F_K$ is a bijection and for each $X \in \text{Dom}(F)$, $|F_K(X)| = |X|$ where $|…|$ denotes length

Random Functions and Permutations

- Let
  - $\text{Func}(l, L)$ denote the set of all functions from $\{0,1\}^l$ to $\{0,1\}^L$
  - $\text{Perm}(l)$ denote the set of all permutations on $\{0,1\}^l$

- Think of $\text{Func}(l, L)$ and $\text{Perm}(l)$ as function families
  - $\text{Keys}(\text{Func}(l, L)) = \{ (Y_1, \ldots, Y_2) : Y_1, \ldots, Y_{2l} \in \{0,1\}^L \}$ with the uniform distribution, where
    $\text{Func}(l, L)((Y_1, \ldots, Y_2), X) = Y_X$, interpreting $X$ as an integer
  - $\text{Keys}(\text{Perm}(l)) = \{ (Y_1, \ldots, Y_2) : Y_1, \ldots, Y_{2l} \in \{0,1\}^l \text{ and are distinct } \}$ with the uniform distribution, where
    $\text{Perm}(l)((Y_1, \ldots, Y_2), X) = Y_X$, interpreting $X$ as an integer

- A random function is $f \leftarrow R \text{ Func}(l, L)$
- A random permutation is $f \leftarrow R \text{ Perm}(l)$
Pseudorandom Functions

A pseudorandom function family is a function family for which the behavior of a random instance is “computationally indistinguishable” from that of a random function.

To define this notion, consider a family

\[ F: \text{Keys}(F) \times D \rightarrow R \]

Now consider a distinguisher who is given “black-box” access (or “oracle access”) to a function \( g \), where \( g \) is either

- Chosen at random from \( F \), i.e., \( g \leftarrow_F F \)
- Chosen at random from \( \text{Func}(D, R) \), i.e., from the set of all functions from \( D \) to \( R \)

By querying \( g \), the distinguisher must determine which of the two possibilities was used to create \( g \).

---

The distinguisher must guess which world it is in

- It outputs “0” or “1” after interacting with this oracle
PRF Definition

Let $F$: $\text{Keys}(F) \times D \to R$ be a family of functions, and let $A$ be an algorithm that takes an oracle for a function $g$: $D \to R$ and returns a bit. We consider two experiments:

Experiment $\text{Expt}_F^{\text{prf}-1}(A)$
\[ K \leftarrow \text{Keys}(F); \]
\[ b \leftarrow A^FK; \]
\[ \text{return } b \]

Experiment $\text{Expt}_F^{\text{prf}-0}(A)$
\[ g \leftarrow \text{Func}(D, R); \]
\[ b \leftarrow A^g; \]
\[ \text{return } b \]

PRF Definition (cont.)

The prf-advantage of $A$ is defined as
\[ \text{Adv}_F^{\text{prf}}(A) = \Pr[\text{Expt}_F^{\text{prf}-1}(A) = 1] - \Pr[\text{Expt}_F^{\text{prf}-0}(A) = 1] \]

For any $t$, $q$, and $\mu$ we define the prf-advantage of $F$ as
\[ \text{Adv}_F^{\text{prf}}(t, q, \mu) = \max_A \left\{ \text{Adv}_F^{\text{prf}}(A) \right\} \]

where the maximum is over all $A$ having time complexity $t$ and making at most $q$ oracle queries, the sum of the lengths of these queries being at most $\mu$ bits.
Peculiarities of This Definition

- **Where is the key length \( k \)?**
  - \( k \) doesn’t directly matter, but rather only the advantage gained by attackers does
  - In a well-designed \( F \) with key length \( k \), the prf-advantage of \( F \) should be something like \( t/2^k \), but this is just an ideal

- **There is no statement of when \( F \) is “secure”**
  - We will deal with this later
  - Informally, \( F \) is “secure enough” if the prf-advantage of \( F \) is “small enough” for “practical” values of the resource parameters

Pseudorandom Permutations

- **When considering permutations, the treatment is largely the same, but with two differences**

- **We replace \( \text{Func}(D, R) \) with \( \text{Perm}(D) \)**

- **There are two types of attacks we can consider**
  - Chosen plaintext attacks: As before, the adversary is given an oracle for the function \( g \) being tested.
  - Chosen ciphertext attacks: The adversary gets, in addition, an oracle for \( g^{-1} \)
**PRPs Under Chosen Plaintext Attack**

- Let $F$: Keys($F$) $\times$ $D$ $\rightarrow$ $D$ be a family of functions, and let $A$ be an algorithm that takes an oracle for a function $g$: $D$ $\rightarrow$ $D$ and returns a bit. We consider two experiments:

  \[
  \begin{align*}
  \text{Experiment } & \text{Expt}^{\text{prp-1}}_F(A) \\
  K \leftarrow & \text{Keys}(F); \\
  b \leftarrow & A^{FK}; \\
  \text{return } b \\
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{Experiment } & \text{Expt}^{\text{prp-0}}_F(A) \\
  g \leftarrow & \text{Perm}(D); \\
  b \leftarrow & A^g; \\
  \text{return } b \\
  \end{align*}
  \]

**PRPs Under Chosen Plaintext Attack**

- The prp-cpa-advantage of $A$ is defined as

  \[
  \text{Adv}^{\text{prp-cpa}}_F(A) = \Pr[\text{Expt}^{\text{prp-cpa-1}}_F(A) = 1] - \Pr[\text{Expt}^{\text{prp-cpa-0}}_F(A) = 1]
  \]

- For any $t$, $q$, and $\mu$ we define the prp-cpa-advantage of $F$ as

  \[
  \text{Adv}^{\text{prp-cpa}}_F(t, q, \mu) = \max_A \{\text{Adv}^{\text{prp-cpa}}_F(A)\}
  \]

  where the maximum is over all $A$ having time complexity $t$ and making at most $q$ oracle queries, the sum of the lengths of these queries being at most $\mu$ bits.
PRPs Under Chosen Ciphertext Attack

Let $F: \text{Keys}(F) \times D \rightarrow D$ be a family of functions, and let $A$ be an algorithm that takes two oracles for functions $g: D \rightarrow D$ and $h: D \rightarrow D$ and returns a bit. We consider two experiments:

<table>
<thead>
<tr>
<th>Experiment $\text{Expt}_{F}^{\text{prp-cca-1}}(A)$</th>
<th>Experiment $\text{Expt}_{F}^{\text{prp-cca-0}}(A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K \leftarrow \text{Keys}(F)$; $b \leftarrow A_{F,K}^{g,F}$; return $b$</td>
<td>$g \leftarrow \text{Perm}(D)$; $b \leftarrow A_{g}^{g^{-1},1}$; return $b$</td>
</tr>
</tbody>
</table>

The prp-cca-advantage of $A$ is defined as

$$\text{Adv}_{F}^{\text{prp-cca}}(A) = \Pr[\text{Expt}_{F}^{\text{prp-cca-1}}(A) = 1] - \Pr[\text{Expt}_{F}^{\text{prp-cca-0}}(A) = 1]$$

For any $t, q_e, \mu_e, q_d, \mu_d$ we define the prp-cca-advantage of $F$ as

$$\text{Adv}_{F}^{\text{prp-cca}}(t, q_e, \mu_e, q_d, \mu_d) = \max_{A} \left\{ \text{Adv}_{F}^{\text{prp-cca}}(A) \right\}$$

where the maximum is over all $A$ having time complexity $t$ and making

- At most $q_e$ oracle queries to the $g$ oracle, the sum of the lengths of these queries being at most $\mu_e$ bits, and
- At most $q_d$ oracle queries to the $g^{-1}$ oracle, the sum of the lengths of these queries being at most $\mu_d$ bits.
The Birthday Problem

Proposition: Let $C(N, q)$ denote the probability of a collision when throwing $q \geq 1$ balls at random into $N \geq q$ buckets. Then,

$$0.3 \cdot \frac{q(q-1)}{N} \leq C(N, q) \leq 0.5 \cdot \frac{q(q-1)}{N} \text{ for } q \leq \sqrt{2N}$$

\begin{align*}
\text{Proof of upper bound:} & \quad \text{Let } C_i \text{ be the event that the } i\text{-th ball collides with one of the previous. Then } Pr[C_i] \leq (i-1)/N. \text{ So,} \\
C(N, q) &= Pr[C_1 \cup C_2 \cup \ldots \cup C_q] \\
&\leq Pr[C_1] + Pr[C_2] + \ldots + Pr[C_q] \\
&\leq \frac{0}{N} + \frac{1}{N} + \ldots + \frac{q-1}{N} \\
&= \frac{q(q-1)}{2N}
\end{align*}

\begin{align*}
\text{Proof of lower bound:} & \quad \text{Fact: If } 0 \leq x \leq 1, \text{ then } \left(1-\frac{1}{e}\right) \cdot x \leq 1 - e^{-x} \leq x. \\
\text{Let } D_i \text{ be the event that there is no collision after throwing } i \text{ balls. Then,} \\
Pr[D_1] &= 1 \\
Pr[D_{i+1} \mid D_i] &= \frac{N-i}{N} = 1 - \frac{i}{N} \\
\text{So, the probability of no collision at the end is} \\
1 - C(N, q) &= Pr[D_q] = \prod_{i=1}^{q-1} Pr[D_{i+1} \mid D_i] \\
&= \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right) \\
&\leq \prod_{i=1}^{q-1} e^{-i/N} = e^{-1/N} \cdot e^{-2/N} \cdot \ldots \cdot e^{-((q-1)/N)} = e^{-q(q-1)/2N}
\end{align*}
The Birthday Problem

Since \( q \leq \sqrt{2N} \), we know that \( q(q-1)/2N \leq 1 \).

So,

\[
C(N, q) \geq 1 - e^{-q(q-1)/2N} \\
\geq \left(1 - \frac{1}{e}\right) \frac{q(q-1)}{2N} \\
\geq 0.3 \cdot \frac{q(q-1)}{N}
\]

Applications of Birthday Problem to PRFs

**Proposition:** Let \( F: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l \) be a family of permutations, and suppose \( 2 \leq q \leq 2^{(l+1)/2} \) and \( t \) is the time for \( q \) computations of \( F \) plus \( O(ql) \). Then

\[
\text{Adv}_F^{\text{prf}}(t, q, ql) \geq 0.3 \cdot \frac{q(q-1)}{2^l}
\]

\( \triangleright \) Proof: Consider an adversary \( A \) with oracle \( g \) who simply invokes \( g \) with \( q \) distinct values, and returns 1 iff all the responses are distinct.

\[
\Pr[\text{Expt}_F^{\text{prf}-1}(A) = 1] = 1 \quad \Pr[\text{Expt}_F^{\text{prf}-0}(A) = 1] = 1 - C(2^l, q)
\]

So,

\[
\text{Adv}_F^{\text{prf}}(A) = \Pr[\text{Expt}_F^{\text{prf}-1}(A) = 1] - \Pr[\text{Expt}_F^{\text{prf}-0}(A) = 1] \\
= 1 - \left(1 - C(2^l, q)\right) \geq 0.3 \cdot \frac{q(q-1)}{2^l}
\]
**PRFs versus PRPs**

- **Proposition:** Suppose $F : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l$ is a family of permutations. Then,

$$\text{Adv}_{F}^{\text{prf}}(t, q, q l) \leq \frac{q(q-1)}{2^{l+1}} + \text{Adv}_{F}^{\text{prp-cpa}}(t, q, q l)$$

for any $r$ and $q$.

- **Proof:** Let $A$ be an adversary that takes an oracle for a function $g : \{0,1\}^l \rightarrow \{0,1\}^l$. We show that

$$\text{Adv}_{F}^{\text{prf}}(A) \leq \frac{q(q-1)}{2^{l+1}} + \text{Adv}_{F}^{\text{prp-cpa}}(A)$$

where $q$ is the number of oracle queries made by $A$.

---

**PRFs versus PRPs**

Let $B$ denote the adversary that first runs $A$ to obtain an output $b$ and then returns $\neg b$, i.e., the complement of $b$.

$$\text{Adv}_{F}^{\text{prf}}(A) = \Pr[\text{Exp}_{F}^{\text{prf}}(A) = 1] - \Pr[\text{Exp}_{F}^{\text{prf}}(A) = 1]$$

$$= (1 - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1]) - (1 - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1])$$

$$= \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1] - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1]$$

$$= \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1] - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1]$$

$$+ \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1] - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1]$$

$$= \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1] - \Pr[\text{Exp}_{F}^{\text{prf}}(B) = 1] + \text{Adv}_{F}^{\text{prp-cpa}}(A)$$
PRFs versus PRPs

So, it suffices to show that

\[ \Pr[\text{Exp}\_F^{\text{prf-o}}(B) = 1] - \Pr[\text{Exp}\_F^{\text{prp-cpa-o}}(B) = 1] \leq \frac{q(g-1)}{2^{l+1}} \]

Let \( g \) denote the oracle in \( \text{Exp}\_F^{\text{prf-o}}(B) \), and assuming (wlog) that all queries by \( B \) to the oracle are distinct, let \( D \) denote the event that all the responses are distinct.

\[
\Pr[\text{Exp}_F^{\text{prf-o}}(B) = 1] = \Pr[B^g = 1] = \Pr[B^g = 1 \mid D] \cdot \Pr[D] + \Pr[B^g = 1 \mid \neg D] \cdot \Pr[\neg D] \\
\leq \Pr[B^g = 1 \mid D] + \Pr[\neg D] \\
= \Pr[\text{Exp}_F^{\text{prp-cpa-o}}(B) = 1] + \Pr[\neg D] \\
\leq \Pr[\text{Exp}_F^{\text{prp-cpa-o}}(B) = 1] + \frac{q(g-1)}{2^{l+1}}
\]

Application: Password Hashing

- An approach for a computer to check a password for user \( U \) is:
  - Store \((U, h(K))\) where \( K \) is the user’s password
  - The user claiming to be \( U \) enters password \( K' \), and the system confirms that \( h(K) \) is stored with \( U \)
  - Here, \( h: \{0,1\}^k \rightarrow \{0,1\}^L \) is a “password hashing function”

- A common choice is \( h(K) = F_K(0^L) \) where \( F \) is a PRP, i.e., a block cipher
  - Is this a good idea?
  - For now, let’s assume that passwords are chosen uniformly at random

- What is the security property we’re trying to achieve?
  - Informally, if an attacker gets the file of \((U, h(K))\) pairs, it shouldn’t be able to learn the password \( K \) ...
  - ... or any other \( K' \) where \( h(K') = h(K) \)
One-Way Functions from PRFs

Let \( h: \{0,1\}^k \to \{0,1\}^L \) be a function, and let \( I \) be an algorithm that on input an \( L \)-bit string returns a \( k \)-bit string. Consider

Experiment \( \text{Exp}_{h}(I) \)

\[
K \leftarrow \{0,1\}^k; \\
y \leftarrow h(K); \\
x \leftarrow I(y); \\
\text{if } h(x) = y \text{ return 1 else return 0}
\]

The owf-advantage of \( I \) is

\[
\text{Adv}_{h}^{\text{owf}}(I) = \Pr[\text{Exp}_{h}(I) = 1]
\]

For any \( t \), we define the owf-advantage of \( h \) as

\[
\text{Adv}_{h}^{\text{owf}}(t) = \max_I \{ \text{Adv}_{h}^{\text{owf}}(I) \}
\]

where the maximum is over all \( I \) having time complexity \( t \).

---

One-Way Functions from PRFs

Intuitively, we’d like to show that if \( h(K) = F_K(0^L) \) where \( F \) is a “secure” PRP, then \( h \) is one-way.

- That is, if \( h \) is not one-way, then \( F \) is not a good PRP.

**Theorem:** Let \( F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L \) be a family of functions, and define \( h(K) = F_K(0^l) \). If \( k < L \), then

\[
\text{Adv}_{h}^{\text{owf}}(t) \leq \frac{1}{1 - 2^{k-L}} \cdot \text{Adv}_{F}^{\text{owf}}(t',1,l)
\]

where \( t' \) is \( t \) plus the time for one computation of \( F \).

- Note: Since \( k < L \),

\[
\frac{1}{1 - 2^{k-L}} \leq 2
\]
One-Way Functions from PRFs

Proof: Let $I$ be any adversary attempting to invert $h$. Define a PRF adversary $D_I$ as follows.

Adversary $D_I$

$y \leftarrow g(0^l)$;

$x \leftarrow I(y)$;

if $F_x(0^l) = y$ return 1 else return 0

For convenience, let

$\epsilon = \text{Adv}^\text{out}_h(I)$

First note that

$\Pr[\text{Expt}^{\text{prf-1}}_F(D_I) = 1] = \epsilon$

since $\text{Expt}^{\text{out}}_h(I)$ and $\text{Expt}^{\text{prf-1}}_h(D_I)$ are identical.

One-Way Functions from PRFs

Second, we show that

$\Pr[\text{Expt}^{\text{prf-0}}_F(D_I) = 1] \leq \frac{2^k}{2^L} \cdot \epsilon$

In $\text{Expt}^{\text{prf-0}}_F(D_I)$, $y$ is uniformly distributed over $\{0,1\}^L$. So, we want to upper bound

$\delta = \Pr[y \leftarrow \mathcal{R}, x \leftarrow I(y) : F_x(0^l) = y]$

where the probability is taken over the choice of $y$ and the random choices made by $I$.

However, here we will prove it only for the case where $I$ is deterministic. (See text for full proof.)
One-Way Functions from PRFs

If $I$ is deterministic, let

$X = \{ x \in \{0,1\}^k : h(I(h(x))) = h(x) \}$

$Y = \{ y \in \{0,1\}^k : h(I(y)) = y \}$

Note that if $y \in Y$ then $I(y) \in X$, and so $|Y| \leq |X|$. So,

$\delta = \frac{|Y|}{2^L}$ by definition of $Y$

$\leq \frac{|X|}{2^L}$ since $|Y| \leq |X|$

$= 2^k \cdot \epsilon$ since $\epsilon = |X|/2^k$

To summarize,

$\Pr[\text{Expt}_{F}^{prf^{-1}}(D_I) = 1] = \epsilon$

$\Pr[\text{Expt}_{F}^{prf^{-0}}(D_I) = 1] \leq \frac{2^k}{2^L} \cdot \epsilon$

So,

$\text{Adv}_{F}^{prf}(D_I) = \Pr[\text{Expt}_{F}^{prf^{-1}}(D_I) = 1] - \Pr[\text{Expt}_{F}^{prf^{-0}}(D_I) = 1]$

$\geq \epsilon - \frac{2^k}{2^L} \cdot \epsilon$

$= (1 - 2^{k-L}) \cdot \epsilon$

And by dividing we get

$\epsilon \leq \frac{1}{1 - 2^{k-L}} \cdot \text{Adv}_{F}^{prf}(D_I)$
Applications of PRFs and PRPs

- PRFs and PRPs are very useful notions in cryptography

- One of their most common uses is to model block ciphers
  - You’ve probably heard of some, like DES and AES
  - PRP is now the most accepted security criteria for a block cipher

- Unfortunately, we do not know how to prove that a block cipher is, in fact, a PRP
  - We try to design it to be, and then assume it is

Building PRFs and PRPs in Practice

- Practical PRF/PRP design has historically be driven by two principles, proposed originally by Shannon in 1949

- Diffusion
  - In many applications, an attacker has knowledge of the statistical characteristics of the inputs
  - Diffusion strives to dissipate the structure of the plaintext into long range statistics of the ciphertext
  - This is achieved by having each plaintext byte affect the value of many ciphertext bytes, or in other words, by having each ciphertext byte depend on many plaintext bytes

- Confusion
  - Seeks to make the relationship between the ciphertext and the key as complex as possible, so that mapping ciphertext statistics to key statistics is difficult
The AES Competition

- In 1999, the U.S. National Institute of Standards and Technology (NIST) issued a call for proposals for a new Advanced Encryption Standard ("AES")
  - Would replace the aging Data Encryption Standard that had been standardized in the late 1970s
  - Block cipher was required to have a block length of 128 bits and support key lengths of 128, 192, and 256 bits
  - Call for proposals yielded over 20, reduced to 15 in a first round and 5 in a second round

- NIST completed its evaluation process in November 2001
  - Settled on an algorithm called “Rijndael”

Differences Between Rijndael and AES

- As originally proposed, Rijndael permitted the block length and key length to be specified independently
  - Could be 128, 192 or 256 bits
- As initially standardized, AES limited the block length to 128 bits
  - Key length could still be chosen as 128, 192 or 256 bits
- This is the only difference between the two

- Our description here will assume that both block length and key length are 128 bits
High-Level Structure

- Input and output are 128-bit blocks
- Consists of 11 rounds:
  - Round 0 consists only of an “add round key” step
  - Rounds 1-9 consist of four steps each
  - Round 10 consists of three steps
- Each round has a step that is dependent on the key
  - Expanded key is denoted by “\(w\)"
  - \(w[4i, 4i+3]\) denote the 4-word (128-bit) round key for round \(i\)

Input, State, Output and Key

- Input, state, and output is a square matrix of bytes
  - Matrix is column-oriented
- Key is also a square matrix, but is expanded to 44 words
  - 4 words per round
Stages

- Most block ciphers utilize at least the following types of operations
  - Substitution: Portions of state are replaced by other values (in a way that can be inverted)
  - Permutation: State is reordered

- AES stages consist of three substitutions and one permutation
  - Substitute bytes: Performs a byte-by-byte substitution of the block
  - Shift rows: A simple permutation
  - Mix columns: A substitution that makes use of arithmetic over GF($2^8$)
  - Add round key: XOR of the current block with a portion of the expanded key

An AES Round
The Substitute Bytes Stage

- This is a simple table lookup
- AES defines a 16x16 table ("S-box") of all 256 8-bit values
- Each byte of the “State” matrix is used to index into the S-box
  - First 4 bits indexes the S-box row, second 4 bits indexes column

S-box Rationale

- S-boxes are designed to withstand known cryptanalytic attacks

- Some goals
  - Low correlation between input bits and output bits
  - Output cannot be described as a simple mathematical function of input
  - S-box has no fixed points (S-box(a) = a) or “opposite” fixed points (S-box(a) = ¬a, where ¬a denotes bitwise complement of a)
  - S-box has no self-inverses (S-box(a) = IS-box(a) where IS-box denotes the inverse of S-box)
S-box Construction

- Initialize $S-box(a) = a$
- Replace $S-box(a)$, $a \neq 0$, with $a^{-1}$ in $GF(2^8)$
  - AES uses irreducible polynomial $f(x) = x^8 + x^4 + x^3 + x + 1$
- Transform each byte $b_7b_6b_5b_4b_3b_2b_1b_0$ as follows, where operations are in $GF(2)$

\[
\begin{array}{cccccccc}
 b_0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & b_0
 \\
b_2 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & b_1
 \\
b_2 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & b_0
 \\
b_3 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & b_3
 \\
b_4 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & b_2
 \\
b_5 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & b_1
 \\
b_6 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & b_0
 \\
b_7 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & b_7
\end{array}
\]

Shift Row Permutation

- A seemingly simple permutation
  - Row $i$, $i \in \{0, \ldots, 3\}$, undergoes a $i$-byte circular left shift
- Ensures that each column is spread across all four columns
  - Significant since input is copied into columns, and since (as we will see) round key is applied to State column-by-column
Mix Column Substitution

- Each byte of each column in State is mapped into a new value that is a function of all four bytes in the column

\[
\begin{bmatrix}
    s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
    s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
    s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
    s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3}
\end{bmatrix}
\begin{bmatrix}
    02 & 03 & 01 & 01 \\
    01 & 02 & 03 & 01 \\
    01 & 01 & 02 & 03 \\
    03 & 01 & 01 & 02
\end{bmatrix}
\begin{bmatrix}
    s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
    s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
    s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
    s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3}
\end{bmatrix}
\]

- Operations are done in $GF(2^8)$
  - Coefficients are written in hexadecimal

Rationale for Mix Column

- Coefficients are based on a linear code with maximal distance between code words
  - Ensures good mixing among the bytes of each column
  - Together with shift row permutation, ensures that all output bits depend on all input bits after a few rounds

- Coefficients were influenced by implementation considerations
  - Intended to minimize the number of XORs in multiplications
  - Inverse coefficients for decryption do not share this property, but encryption is more important
    - Some encryption modes (e.g., counter mode) do not require inverting
    - Some applications (e.g., MACs) do not require inverting
Add Round Key Stage

- State is bitwise XOR'd with the 128-bit round key

Rationale
- As simple as possible
- Affects every bit of State

AES Key Expansion

- Input is a 4-word (16 byte) key
- Output is an array of 44 words
- First 4 output words is input key
- Remainder are generated 4 words at a time
- Utilizes a function $g$ as shown for every fourth word
AES Key Expansion

- **Function $g$ utilizes the following**
  - Function RotWord performs a one-byte circular left-shift on a word
    \[ [B_0, B_1, B_2, B_3] \rightarrow [B_1, B_2, B_3, B_0] \]
  - Function SubWord performs a byte substitution on each byte of its input word, using the S-box
  - A round constant $Rcon[j]$ of length one word

- **Then, $g$ in round $j$ is defined as follows**

  \[
g(w) = \text{SubWord}(\text{RotWord}(w)) \oplus Rcon[j]
  \]
AES Key Expansion

**Rationale**
- Designed to be resistant to known cryptanalytic attacks
- Inclusion of round-dependent constant eliminates symmetry in round key generation across rounds
- Designed to be fast on a range of processors
- Diffusion of cipher key differences into round keys
  - Each key bit affects many round key bits
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only

---

Implementation on 32-bit Processors

- Denote State matrix elements by $a_{ij}$, and round key matrix elements by $k_{ij}$

- Recall that stages take the following form

<table>
<thead>
<tr>
<th>SubBytes</th>
<th>$b_{i,j} \leftarrow S[a_{i,j}]$</th>
<th>MixColumns</th>
<th>$d_{0,j}$</th>
<th>$d_{1,j}$</th>
<th>$d_{2,j}$</th>
<th>$d_{3,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>01</td>
<td>$c_{0,j}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>$c_{1,j}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>03</td>
<td>01</td>
<td>01</td>
<td>02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ShiftRows</th>
<th>$c_{0,j}$</th>
<th>$c_{1,j}$</th>
<th>$c_{2,j}$</th>
<th>$c_{3,j}$</th>
<th>$b_{0,j}$</th>
<th>$b_{1,j}$</th>
<th>$b_{2,j}$</th>
<th>$b_{3,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{0,j}$</td>
<td>$c_{1,j}$</td>
<td>$c_{2,j}$</td>
<td>$c_{3,j}$</td>
<td>$b_{0,j}$</td>
<td>$b_{1,j}$</td>
<td>$b_{2,j}$</td>
<td>$b_{3,j}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AddRoundKey</th>
<th>$a_{0,j}$</th>
<th>$a_{1,j}$</th>
<th>$a_{2,j}$</th>
<th>$a_{3,j}$</th>
<th>$d_{0,j}$</th>
<th>$d_{1,j}$</th>
<th>$d_{2,j}$</th>
<th>$d_{3,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{0,j}$</td>
<td>$a_{1,j}$</td>
<td>$a_{2,j}$</td>
<td>$a_{3,j}$</td>
<td>$d_{0,j}$</td>
<td>$d_{1,j}$</td>
<td>$d_{2,j}$</td>
<td>$d_{3,j}$</td>
<td>$k_{0,j}$</td>
</tr>
</tbody>
</table>

---

Copyright © 2017 by Michael Reiter
All rights reserved.
Implementation on 32-bit Processors

- If we combine these we get

\[
\begin{bmatrix}
  e_{0,j} \\
  e_{1,j} \\
  e_{2,j} \\
  e_{3,j}
\end{bmatrix}
= \begin{bmatrix}
  02 & 03 & 01 & 01 \\
  01 & 02 & 03 & 01 \\
  01 & 01 & 02 & 03 \\
  03 & 01 & 01 & 02
\end{bmatrix}
\begin{bmatrix}
  S[a_{0,j}] \\
  S[a_{1,j-1}] \\
  S[a_{2,j-2}] \\
  S[a_{3,j-3}]
\end{bmatrix}
\oplus
\begin{bmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{bmatrix}
\]

or

\[
\begin{bmatrix}
  e_{0,j} \\
  e_{1,j} \\
  e_{2,j} \\
  e_{3,j}
\end{bmatrix}
\leftarrow \begin{bmatrix}
  02 & 03 & 01 & 01 \\
  01 & 02 & 03 & 01 \\
  01 & 01 & 02 & 03 \\
  03 & 01 & 01 & 02
\end{bmatrix}
\left(\begin{bmatrix}
  S[a_{0,j}] \\
  S[a_{1,j-1}] \\
  S[a_{2,j-2}] \\
  S[a_{3,j-3}]
\end{bmatrix} + \begin{bmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{bmatrix}\right)
\]

Implementation on 32-bit Processors

- Suppose we now compute 4 tables in advance, each mapping a byte to a 32-bit word
  - Each table consumes 1KB

\[
T_0(x) = \begin{bmatrix}
  02 \\
  01 \\
  03
\end{bmatrix} \cdot S[x]
\]

\[
T_1(x) = \begin{bmatrix}
  03 \\
  02 \\
  01
\end{bmatrix} \cdot S[x]
\]

\[
T_2(x) = \begin{bmatrix}
  01 \\
  03 \\
  02
\end{bmatrix} \cdot S[x]
\]

\[
T_3(x) = \begin{bmatrix}
  01 \\
  01 \\
  03
\end{bmatrix} \cdot S[x]
\]

Copyright © 2017 by Michael Reiter
All rights reserved.
Implementation on 32-bit Processors

- Then, we can define a round function operating on a column, as follows

\[
\begin{bmatrix}
  s_{0,j} \\
  s_{1,j} \\
  s_{2,j} \\
  s_{3,j}
\end{bmatrix} \leftarrow T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus
\begin{bmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{bmatrix}
\]

- This implementation requires only four table lookups and four XORs per column per round
  - Plus 4KB to store tables