COMP547 Claude Crépeau

INTRODUCTION TO
MODERN
CRYPTOGRAPHY
Second Edition
Jonathan Katz •Yehuda Lindell

Chapter 6 : Practical Constructions of Symmetric-Key Primitives

Practical Constructions of Symmetric-Key Primitives

6.2 Practical Constructions of Pseudorandom Permutations (Block Ciphers)

The constructions of block ciphers that we will explore in this chapter are heuristic, at least in the sense that they have no known proof of security based on any weaker assumption.

Nevertheless, a number of the block ciphers that are used in practice have withstood many years of public scrutiny and attempted cryptanalysis.

It is quite reasonable to assume that these block ciphers are (strong) peudorandom permutations.

6.2 Practical Constructions of Pseudorandom Permutations (Block Ciphers)

The requirement that a certain problem (i.e., factoring) be hard to solve seems “easier to satisfy” than the requirement that a given keyed function be indistinguishable from a random function.

Less important but still relevant differences between the assumptions are that the problem of factoring has been studied much longer than the problem of distinguishing **DES** from a random function.

6.2 Practical Constructions of Pseudorandom Permutations (Block Ciphers)

- The fact that factoring was recognized as a hard mathematical problem well before the advent of cryptographic schemes based on it.

- Most of the cryptanalytic effort directed at **DES** and other block ciphers has focused on *key-recovery attacks*, where the goal is to recover the key $\boldsymbol{k}$ given multiple pairs $\mathbf{(}\boldsymbol{x}, \mathbf{DES}_{\boldsymbol{k}}\mathbf{(}\boldsymbol{x}\mathbf{))}$.

Block Ciphers as Strong Pseudorandom Perm's

- The view that block ciphers should be modelled as pseudorandom permutations has, at least in the recent past, served as a major influence in their design.

- As an example, the call for proposals for the recent Advanced Encryption Standard (**AES**) that we will encounter later in this chapter stated the following evaluation criteria:

Block Ciphers as Strong Pseudorandom Perm's

- *The security provided by an algorithm is the most important factor. . . . Algorithms will be judged on the following factors. . .*

- *The extent to which the algorithm output is indistinguishable from a random permutation on the input block.*

Block Ciphers as Strong Pseudorandom Perm's

- Essentially, this states that a block cipher should be a *pseudorandom permutation*.
- It is unclear to what extent submitted proposals were evaluated as *strong* pseudorandom permutations.
- Had an attack been demonstrated showing that some proposal did not satisfy this criterion, it is unlikely the proposal would have been adopted.

Attacks on Block Ciphers

- *Ciphertext-only attacks*, where the attacker is given only outputs $\{F_k(x_i)\}$ for some unknown inputs $\{x_i\}$.
- *Known-plaintext attacks*, where the attacker is given pairs of inputs and outputs $\{(x_i, F_k(x_i))\}$.
- *Chosen-plaintext attacks*, where the attacker is given $\{(x_i, F_k(x_i))\}$ for inputs $\{x_i\}$ chosen by the attacker.
- *Chosen-ciphertext attacks*, where the attacker is given $\{(x_i, F_k(x_i))\}$ and $\{(F_k^{-1}(y_i), y_i)\}$ for $\{x_i\}$, $\{y_i\}$ chosen by the attacker.

Attacks on Block Ciphers

- This is interpreted very strictly in the context of block ciphers, and a block cipher is generally only considered "good" if the best known attack has time complexity roughly equivalent to a brute-force search for the key.
- If a cipher with key length $\boldsymbol{n} = \mathbf{112}$ can be broken in time $\mathbf{2^{56}}$ (we will see such an example later), the cipher is (generally) considered insecure even though $\mathbf{2^{56}}$ is still a relatively large number.

6.2.1 Substitution-Permutation Networks

- The main property required of a block cipher is that it should behave like a random permutation.
- Of course, a truly random permutation would be too big to describe.
- We need to somehow construct a *concise* function that behaves like a random one.

The confusion-diffusion paradigm

In addition to his work on perfect secrecy, Claude Shannon introduced a basic paradigm for constructing concise random-looking permutations.

The confusion-diffusion paradigm

The basic idea is to construct a random-looking permutation ***F*** with a large block length from many smaller random-looking permutations {$\boldsymbol{f_i}$} having a small block length.

The confusion-diffusion paradigm

Say we want $\boldsymbol{F}$ to have a block length of **128** bits. Define $\boldsymbol{F}$ as follows:

- The key $\boldsymbol{k}$ for $\boldsymbol{F}$ will specify **16** *random* permutations $\boldsymbol{f_1,\ldots,f_{16}}$ that each have an **8**-bit block length.

- Given an input $\boldsymbol{x} \in \{\mathbf{0,1}\}^{\mathbf{128}}$, we parse it as **16** consecutive **8**-bit blocks $\boldsymbol{x_1} \cdots \boldsymbol{x_{16}}$ and then set

$$\boldsymbol{F_k(x) \coloneqq f_1(x_1)\cdots f_{16}(x_{16})}.$$

- (Informally) these $\{\boldsymbol{f}\}$ introduce <u>*confusion*</u> into $\boldsymbol{F}$.

The confusion-diffusion paradigm

- First, a *diffusion* step is introduced whereby the bits of the output are permuted or "mixed".
- Second, the confusion/diffusion steps — together called a *round* — are repeated multiple times.

The confusion-diffusion paradigm

As an example, a two-round block cipher would operate as follows:

First, $\boldsymbol{x'} := \boldsymbol{F_k(x)}$ would be computed.

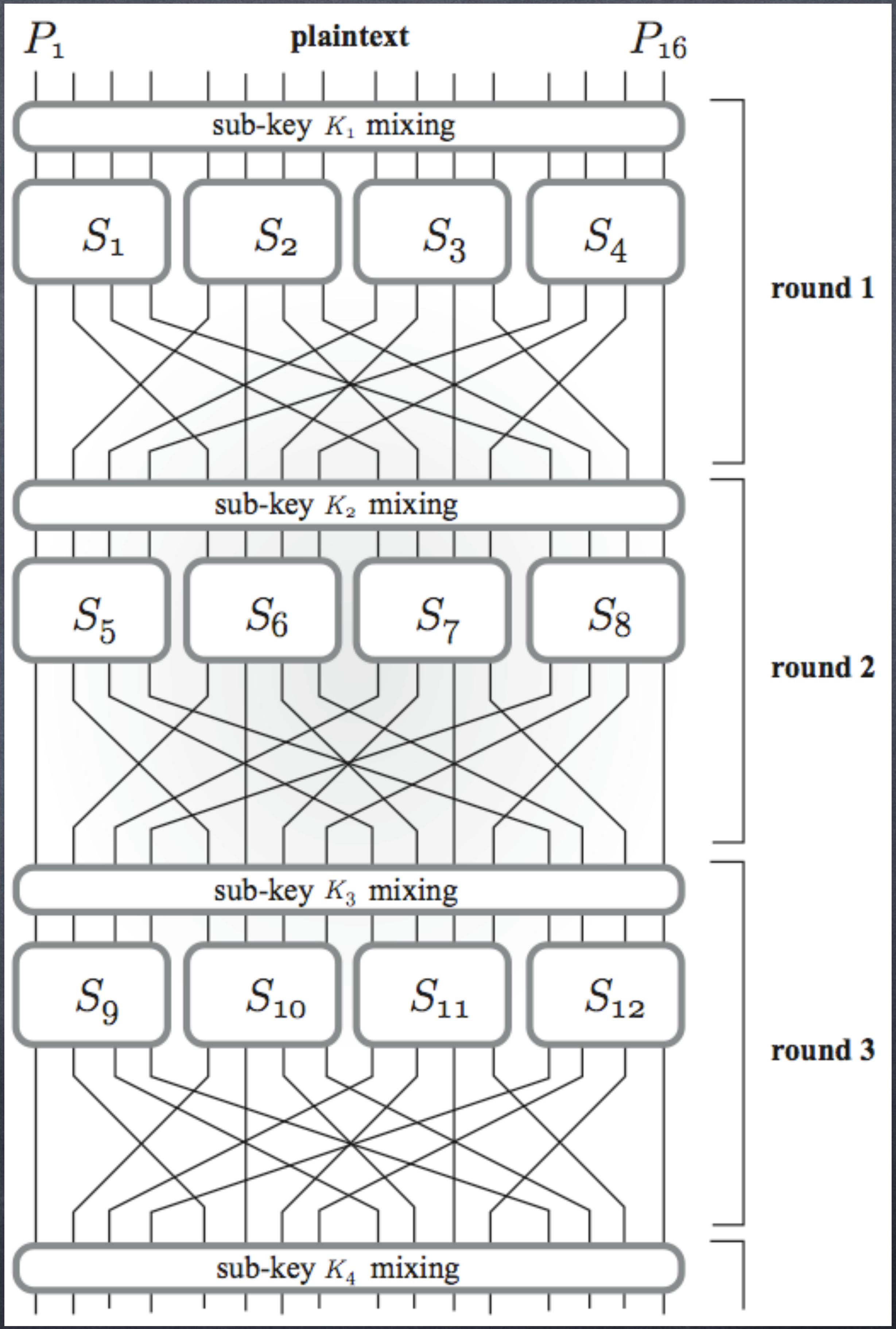
Then the bits of $\boldsymbol{x'}$ would be re-ordered to give $\boldsymbol{x_1}$.

Then $\boldsymbol{x_1'} := \boldsymbol{F_k(x_1)}$ would be computed, and the bits of $\boldsymbol{x_1'}$ would be re-ordered to give the output $\boldsymbol{x_2}$.

The confusion-diffusion paradigm

- Repeated use of confusion and diffusion ensures that any small change in the input will be mixed throughout and propagated to all the bits of the output.
- The effect is that small changes to the input have a significant effect on the output, as one would expect of a random permutation.

Substitution-Permutation Network



Substitution-Permutation Network

- A substitution-permutation network can be viewed as a direct implementation of the confusion-diffusion paradigm
- The main difference here is that we view the round functions $\{f_i\}$ as being *fixed*, and the key is used for a different purpose.
- We now refer to the $\{f_i\}$ as ***S**-boxes*.

Substitution-Permutation Network

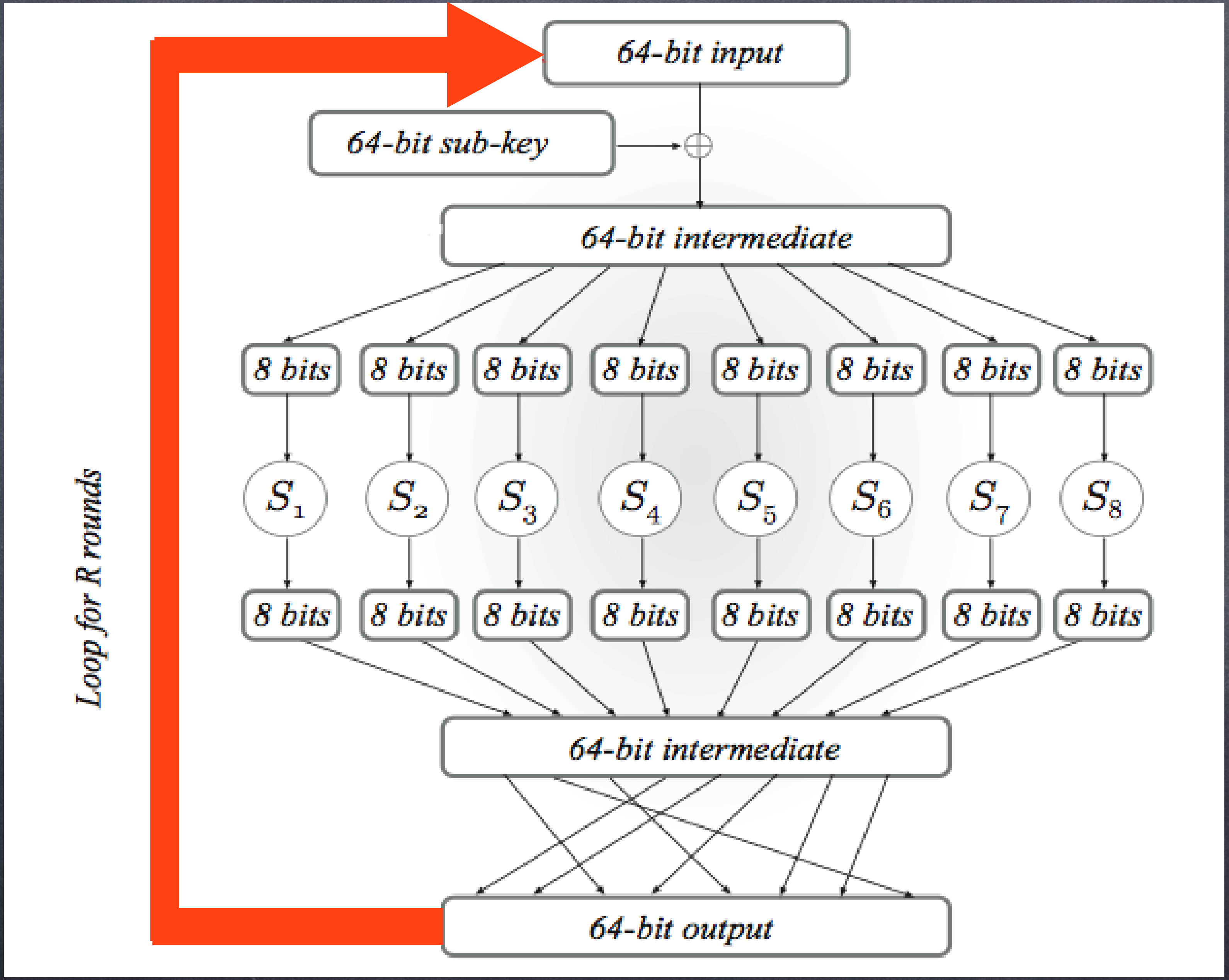
- A substitution-permutation network essentially follows the steps of the confusion-diffusion paradigm outlined earlier.
- Since the ***S***-boxes no longer depend on the key, we need to introduce dependence in some other way.

(In accordance with Kerckhoffs' principle, we assume that the exact structure of the ***S***-boxes and the mixing permutations are publicly-known, with the only secret being the key.)

Substitution-Permutation Network

- There are many ways this can be done.
- We will focus here on the case where this is done by simply **XOR**ing some function of the key with the intermediate results that are fed as input to each round of the network.

Substitution-Permutation Network



Substitution-Permutation Network

- The key to the block cipher is sometimes referred to as the *master key*.
- The sub-keys that are **XOR**ed with the intermediate results in each round are derived from the master key according to a *key schedule*.
- Key schedule is often very simple and may work by just taking subsets of the bits of the key, though more complex schedules can also be defined.

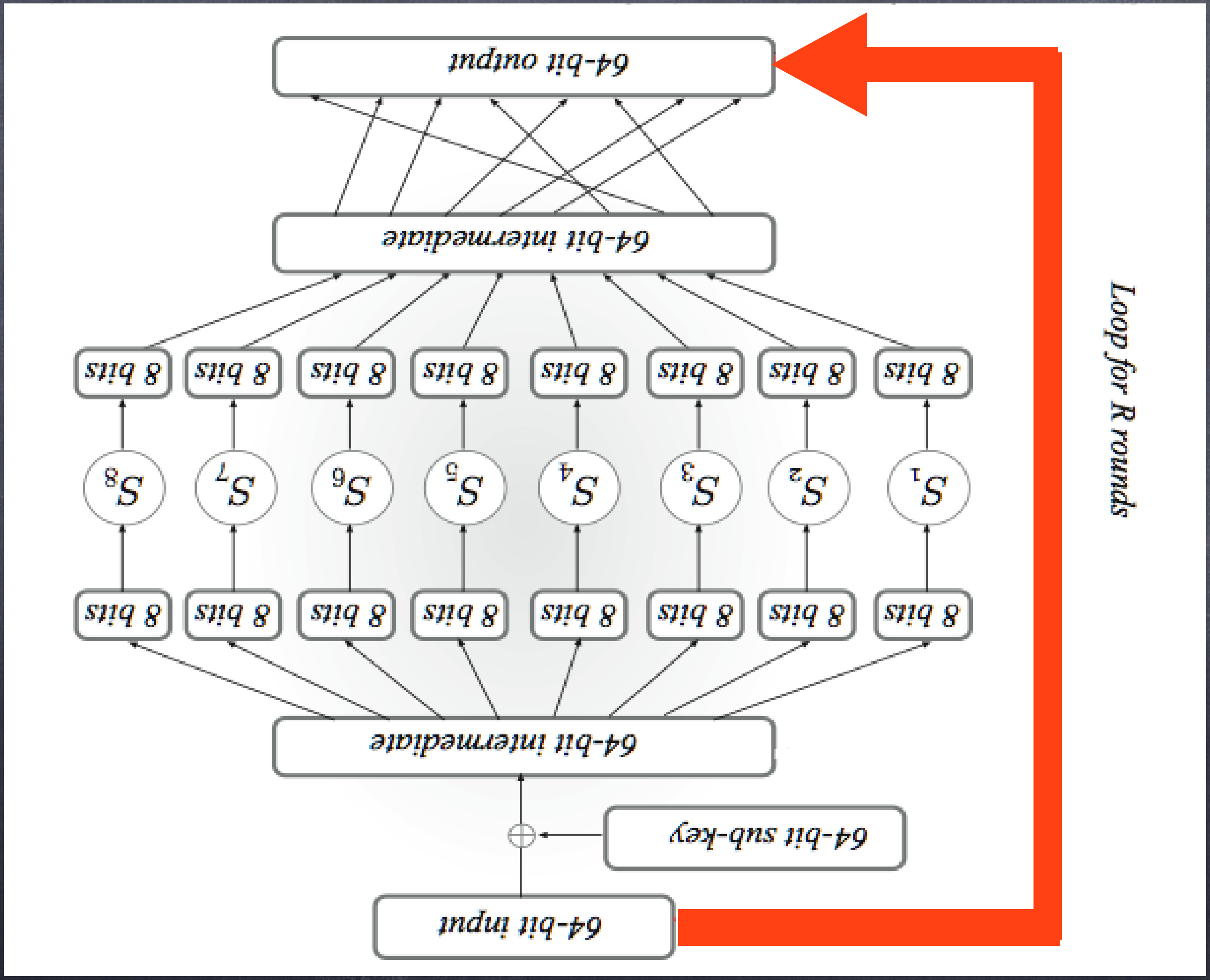
Design principle 1 — invertibility of the S -boxes

- In a substitution-permutation network, the ***S***-boxes must be invertible; that is, they must be one-to-one and onto functions.
- The reason for this is that otherwise the block cipher will not be a permutation.
- To see that making the ***S***-boxes one-to-one and onto suffices, we show that when this holds it is possible to fully determine the input given the output and the key.

Invertibility of the S -boxes

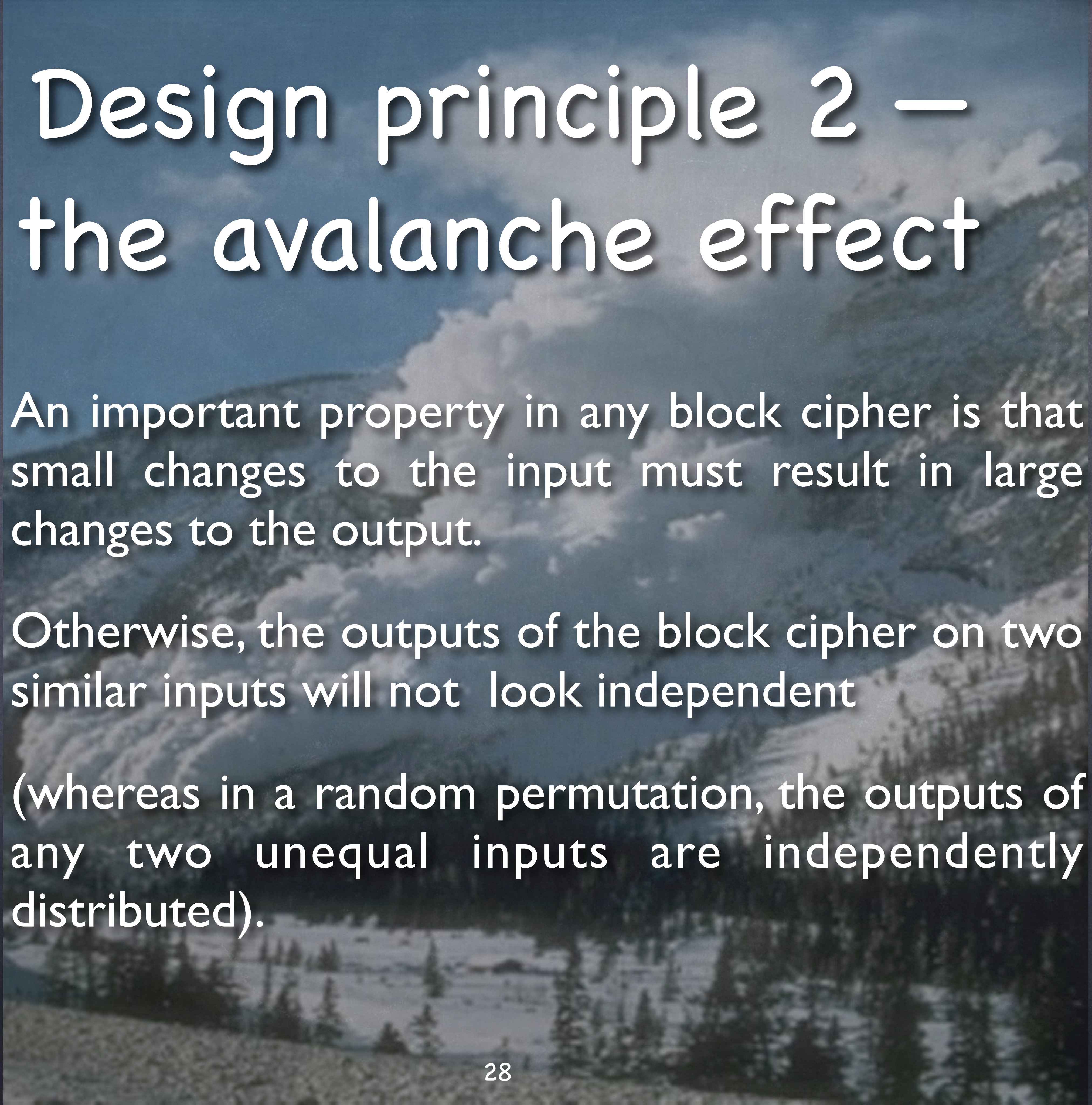
- Specifically, we show that every round can be inverted.
- This implies that the entire network can be inverted by working from the end back to the beginning.

Invertibility of the S -boxes



Invertibility of the S -boxes

PROPOSITION 6.3 *Let* $\mathbf{F}$ *be a keyed function defined by a substitution-permutation network in which the* $\mathbf{S}$*-boxes are all one-to-one and onto. Then regardless of the key schedule and the number of rounds,* $\mathbf{F_k}$ *is a permutation for any choice of* $\mathbf{k}$*.*



Design principle 2 — the avalanche effect

- An important property in any block cipher is that small changes to the input must result in large changes to the output.
- Otherwise, the outputs of the block cipher on two similar inputs will not look independent
- (whereas in a random permutation, the outputs of any two unequal inputs are independently distributed).

the avalanche effect

- Block ciphers are designed to exhibit the *avalanche effect*, meaning that changing a single bit of the input affects every bit of the output.
- (This does not mean that changing one bit of the input *changes* every bit of the output, only that it has some effect on every bit of the output.)

the avalanche effect

It is easy to demonstrate that the avalanche effect holds in a substitution-permutation network provided that the following two properties hold (and sufficiently-many rounds are used):

1. *The **S**-boxes are designed so that changing a single bit of the input to an **S**-box changes at least* two bits *in the output of the **S**-box.*
2. *The mixing permutations are designed so that the output bits of any given **S**-box are spread into* different ***S**-boxes in the next round.*

the avalanche effect

Consider now what happens when the block cipher is applied to two inputs that differ by only a single bit:

1. After the first round, the intermediate values differ in exactly two bit-positions.

This is because **XOR**ing the current sub-key maintains the **1**-bit difference in the intermediate values, and so the inputs to all the ***S***-boxes except one are identical.

In the one ***S***-box where the inputs differ, the output of the ***S***-box causes a **2**-bit difference. The mixing permutation applied to the results changes the positions of these differences, but maintains a **2**-bit difference.

the avalanche effect

2. By the second property mentioned earlier, the mixing permutation applied at the end of the first round spreads the two bit-positions where the intermediate results differ into two *different* ***S***-boxes in the second round.

So, in the second round there are now *two* ***S***-boxes that receive inputs differing by a single bit.

Following the same argument as before, we see that at the end of the second round the intermediate values differ in **4** bits.

the avalanche effect

One might expect that the "best" way to design ***S***-boxes would be to choose them at random.

Interestingly, this turns out not to be the case, at least if we want to satisfy the above criterion.

For example, consider the case of an ***S***-box operating on **4**-bit inputs and let $\boldsymbol{x}$ and $\boldsymbol{x'}$ be two different inputs.

Let $\boldsymbol{y} \coloneqq \boldsymbol{S(x)}$, and now consider choosing $\boldsymbol{y'} \neq \boldsymbol{y}$ at random as the value of $\boldsymbol{S(x')}$.

the avalanche effect

- There are **4** strings that differ from **y** in only **1** bit, and so with probability **4/15** we will choose **y′** that does *not* differ from **y** in two or more bits.
- The problem is compounded when we consider all inputs, and becomes even worse when we consider that multiple ***S***-boxes are needed.
- We conclude based on this example that, as a general rule, it is best to carefully design ***S***-boxed with certain desired properties (in addition to the one discussed above) rather than choosing them blindly at random.

Security of Substitution-Permutation Networks

Experience, along with many years of cryptanalytic effort, indicate that substitution-permutation networks are a good choice for constructing pseudo-random permutations as long as great care is taken in the choice of the ***S***-boxes, the mixing permutations, and the key schedule.

The Advanced Encryption Standard (**AES**), described in **Section 5.5**, is similar in structure to the substitution-permutation network described above, and is widely believed to be a very strong pseudorandom permutation.

Security of Substitution-Permutation Networks

It is important to understand, however, that the strength of a cipher constructed in this way depends heavily on the number of rounds used.

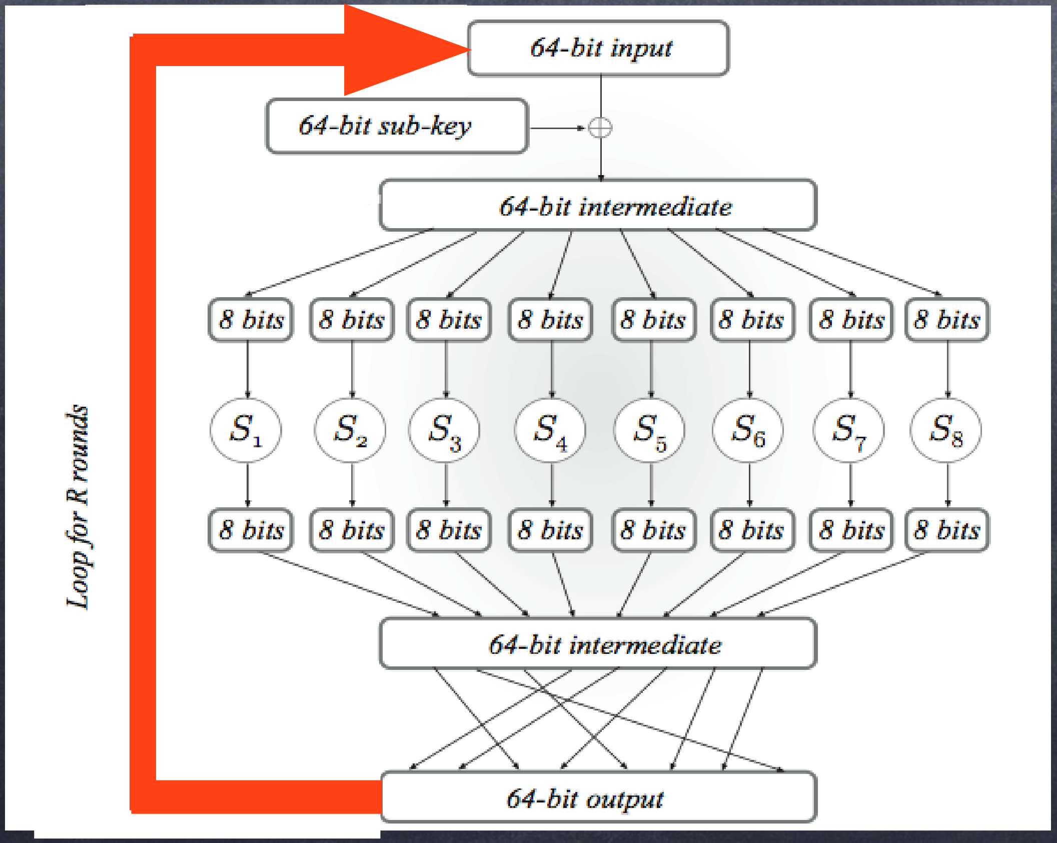
Attacks on reduced-round Subs-Per Network

Attack on a single-round substitution-permutation network: Let ***F*** be a single-round substitution-permutation network.

- We demonstrate an attack where the adversary is given only a *single* input/output pair **(*x*,*y*)** for a randomly-chosen input value ***x***, and easily learns the secret key ***k*** for which $\boldsymbol{y = F_k(x)}$.

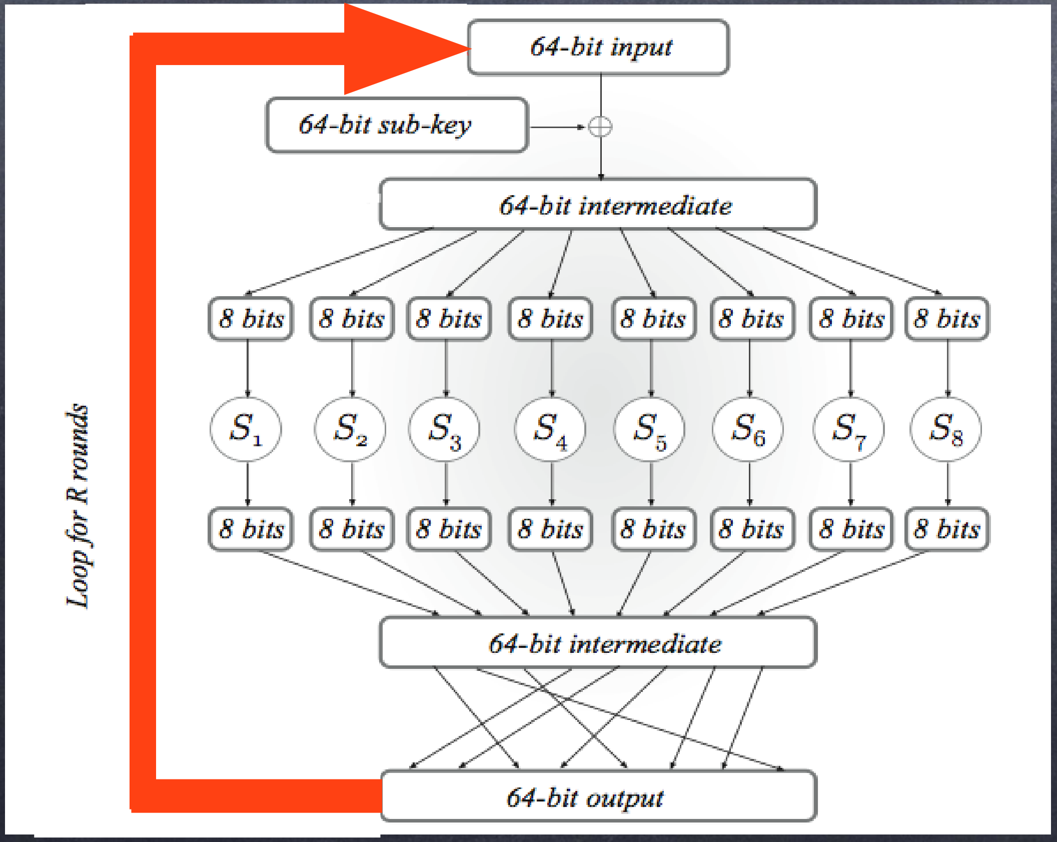
Attacks on reduced-round Subs-Per Network

- The adversary begins with the output value ***y*** and then inverts the mixing permutation and the ***S***-boxes. It can do this because the specification of the permutation and the ***S***-boxes is public.



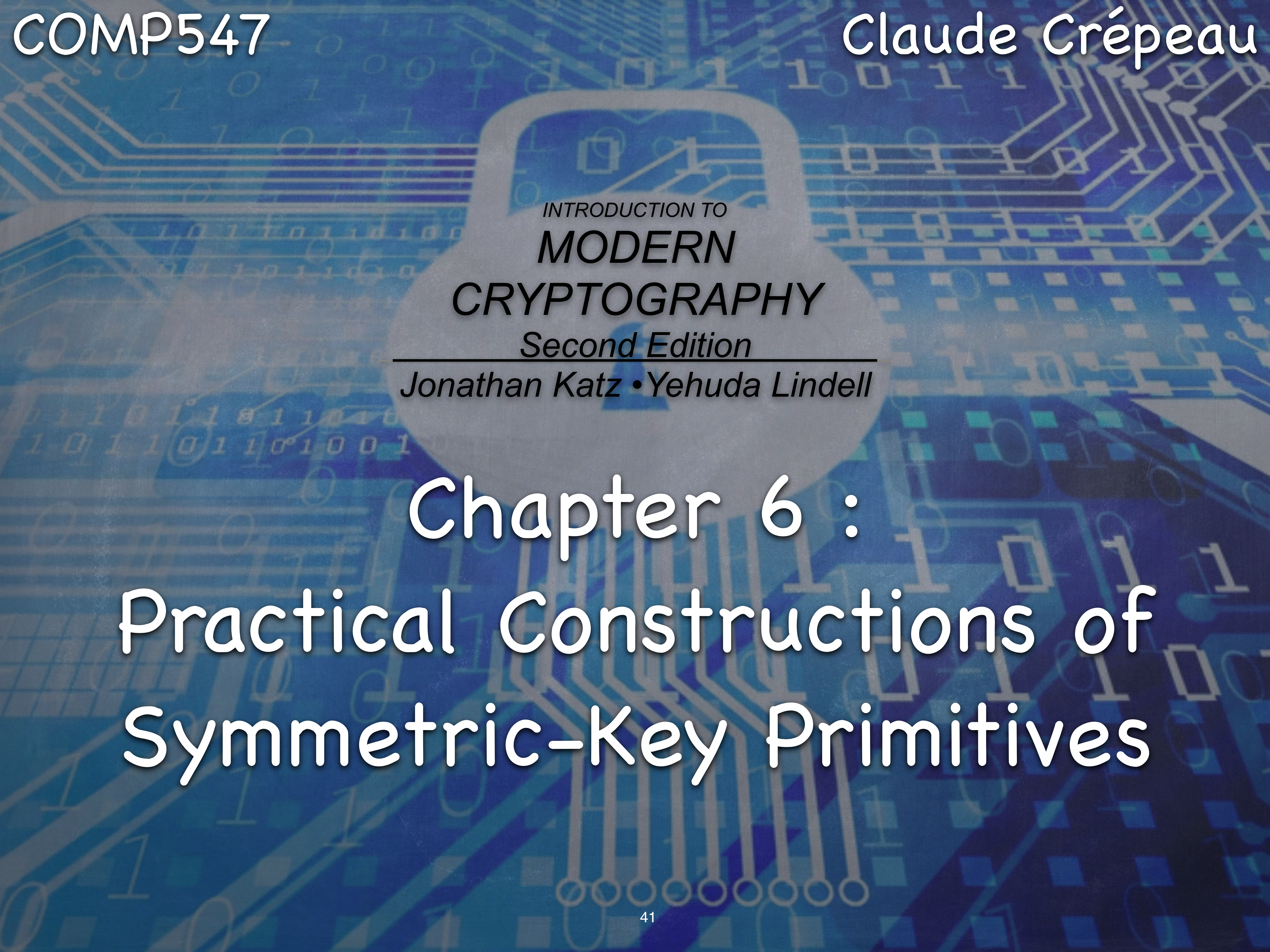
Attacks on reduced-round Subs-Per Network

- The intermediate value that the adversary computes from these inversions is exactly $\boldsymbol{x} \oplus \boldsymbol{k}$ (assuming, without loss of generality, that the master key is used as the sub-key in the only round of the network).



Attack on a single-round Subs-Per Network

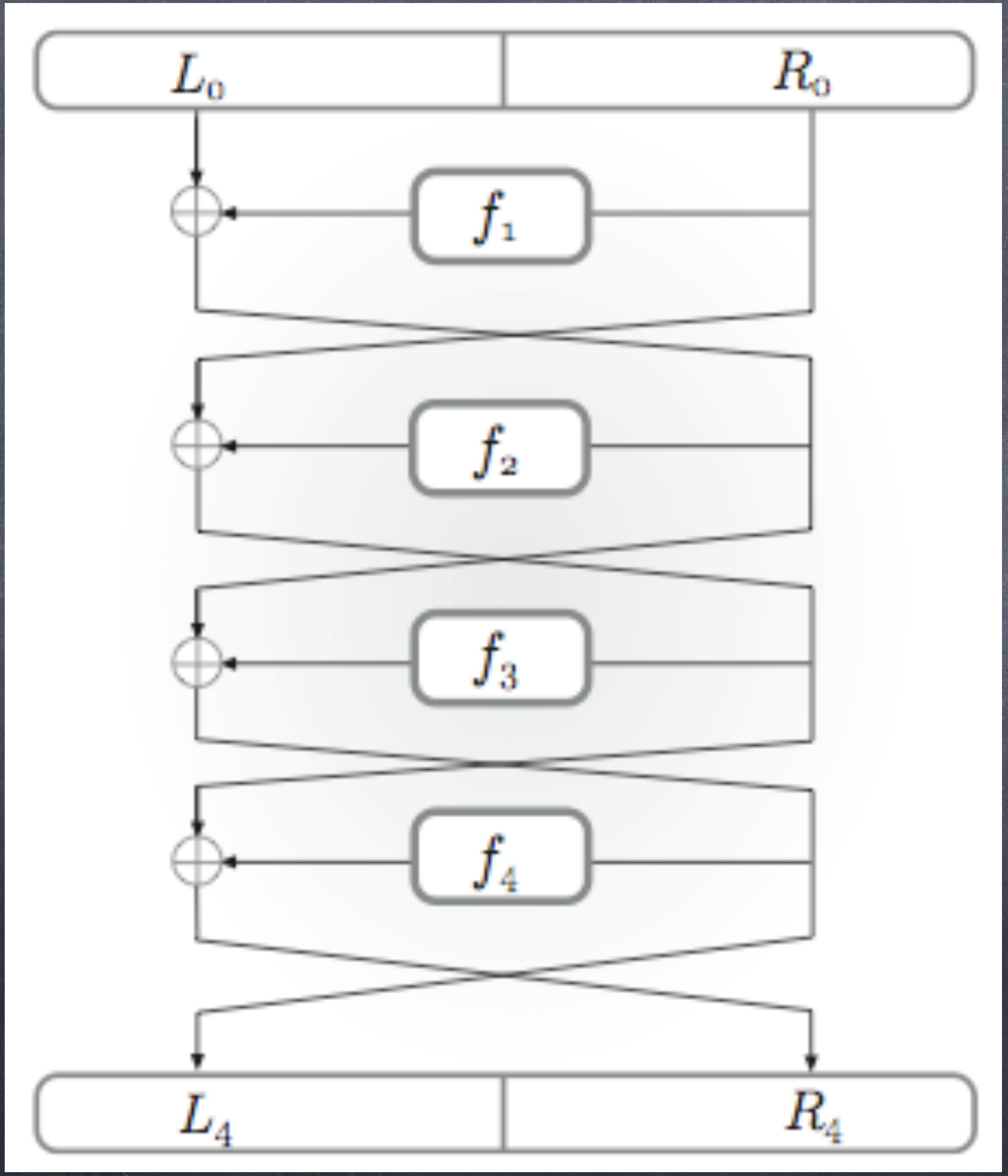
- Since the adversary also has the input ***x***, it immediately derives the secret key ***k***.
- This is therefore a complete break.



Chapter 6 : Practical Constructions of Symmetric-Key Primitives

6.2.2 Feistel Networks

Horst Feistel



Feistel Networks

A Feistel network is an alternative approach for constructing a block cipher.

The low-level building blocks (***S***-boxes, mixing permutations, and a key schedule) are the same.

The difference is in the high-level design.

Feistel Networks

A Feistel network is a way of constructing an invertible function from non-invertible components.

Feistel Networks

- The advantage of Feistel networks over substitution-permutation networks is that a Feistel network eliminates the requirement that **S**-boxes be invertible.
- This is important because a good block cipher should have "unstructured" behavior.
- however, requiring that all the components of the construction be invertible inherently introduces structure.

Feistel Networks

In a Feistel network, round functions need not be invertible.

Round functions typically contain components like **S**-boxes and mixing permutations.

But a Feistel network can deal with any round functions irrespective of their design. When the round functions are constructed from **S**-boxes.

Feistel Networks

The i^{th} round of a Feistel network operates as follows.

The input to the round is divided into two halves denoted L_{i-1} and R_{i-1}.

If the block length of the cipher is n bits, then L_{i-1} and R_{i-1} each have length $^n/_2$.

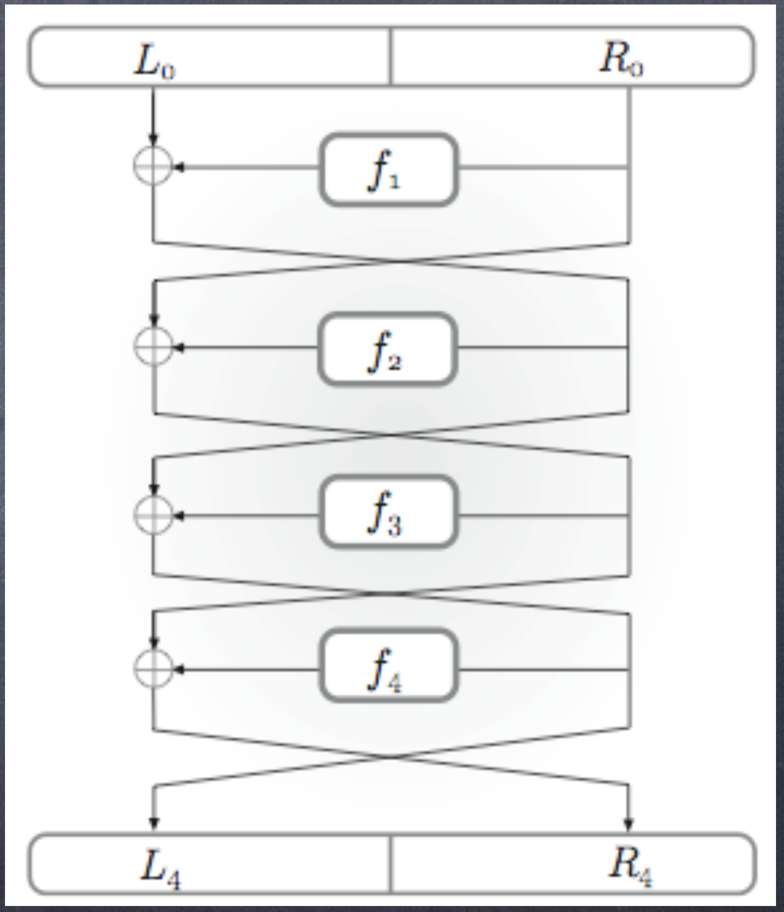
The i^{th} round function f_i will take an $^n/_2$-bit input and produce an $^n/_2$-bit output.

Feistel Networks

The output ($\boldsymbol{L_i}$,$\boldsymbol{R_i}$) of the round is given by

$\boldsymbol{L_i} \coloneqq \boldsymbol{R_{i-1}}$ and $\boldsymbol{R_i} \coloneqq \boldsymbol{L_{i-1}} \oplus \boldsymbol{f_i} (\boldsymbol{R_{i-1}})$.

In a ***t***-round Feistel network, the ***n***-bit input to the network is parsed as **(L_0,R_0)**, and the output is the *n*-bit value **(L_t,R_t)** obtained after applying all ***t*** rounds.



Feistel Networks

- The master key ***k*** is used to derive sub-keys that are used in each round.
- The $\boldsymbol{i}^{\text{th}}$ round function $\boldsymbol{f_i}$ depends on the $\boldsymbol{i}^{\text{th}}$ sub-key, denoted $\boldsymbol{k_i}$.

Feistel Networks

- Formally, the design of a Feistel network specifies a publicly-known *<u>mangler</u> function* f_i' associated with each round i.
- This function f_i' takes as input a sub-key k_i and an $n/2$-bit string and outputs an $n/2$-bit string.
- When the master key is fixed the ith round function f_i is defined via $f_i(R) \overset{\text{def}}{=} f_i'(k_i, R)$.

Inverting a Feistel network

A Feistel network is invertible *regardless of the round functions* $\{f_i\}$ (and of the mangler functions $\{f_i'\}$).

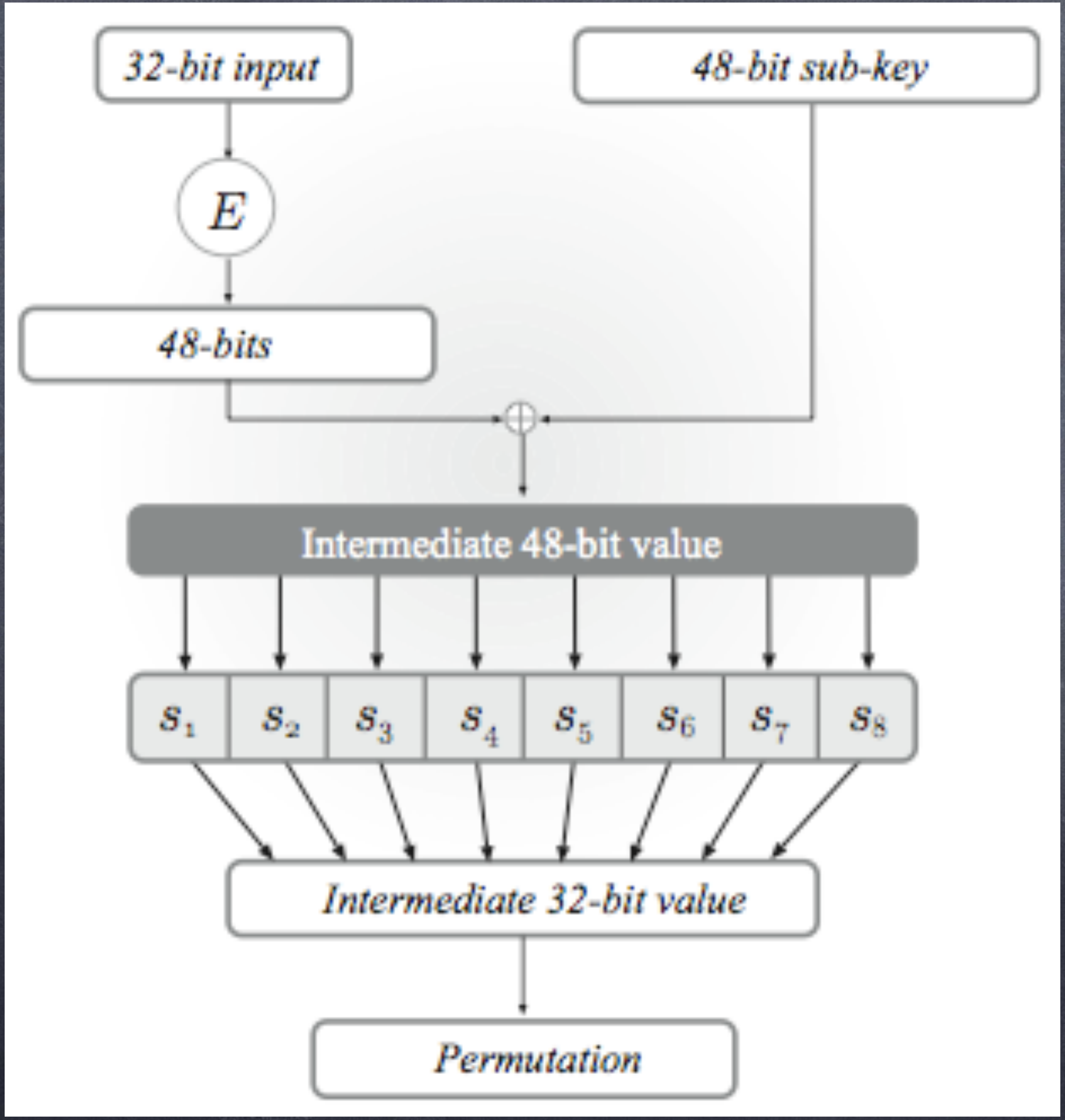
Given the output (L_i, R_i) of the ith round, we can compute (L_{i-1}, R_{i-1}) as follows:

- First set $R_{i-1} \coloneqq L_i$.
- Then compute $L_{i-1} \coloneqq R_i \oplus f_i\,(R_{i-1})$.

Inverting a Feistel network

PROPOSITION 5.2 *Let* $\mathbf{F}$ *be a keyed function defined by a Feistel network.*
Then regardless of the mangler functions $\{\mathbf{f_i'}\}$ *and the number of rounds,* $\mathbf{F_k}$ *is a permutation for any choice of* $\mathbf{k}$.

6.2.3 DES – The Data Encryption Standard


32-bit input
48-bit sub-key
E
48-bits
Intermediate 48-bit value
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
Intermediate 32-bit value
Permutation

The Data Encryption Standard

The Data Encryption Standard, or **DES**, was developed in the 1970s at **IBM** (with help from the National Security Agency)

Adopted in 1977 as a Federal Information Processing Standard (**FIPS**) for the **US**.

In its basic form, **DES** is no longer considered secure due to its short key length of **56** bits.

Nevertheless, it remains in wide use today in its strengthened form of triple-**DES**.

The Data Encryption Standard

- **DES** is of great historical significance, and has undergone intensive scrutiny within the cryptographic community, arguably more than any other cryptographic algorithm in history.
- The common consensus is that, relative to its key length, **DES** is extremely secure.
- Indeed, even after so many years, the best known attack on **DES** *in practice* is a brute-force search over all $\mathbf{2^{56}}$ possible keys.

The Design of DES

The **DES** block cipher is a **16**-round Feistel network with a block length of **64** bits and a key length of **56** bits.

The input/output length of a **DES** round function is **32** bits.

The round functions used in each of the **16** rounds of **DES** are all derived from the same mangler function $f'_i = f'$.

The Design of DES

The *key schedule* of **DES** is used to derive a **48**-bit sub-key $\boldsymbol{k_i}$ for each round from the **56**-bit master key $\boldsymbol{k}$.

The $\boldsymbol{i}$th round function $\boldsymbol{f_i}$ is defined as

$$f_i(R) \overset{\text{def}}{=} f'(k_i, R).$$

The round functions are *non-invertible*.

The Design of DES

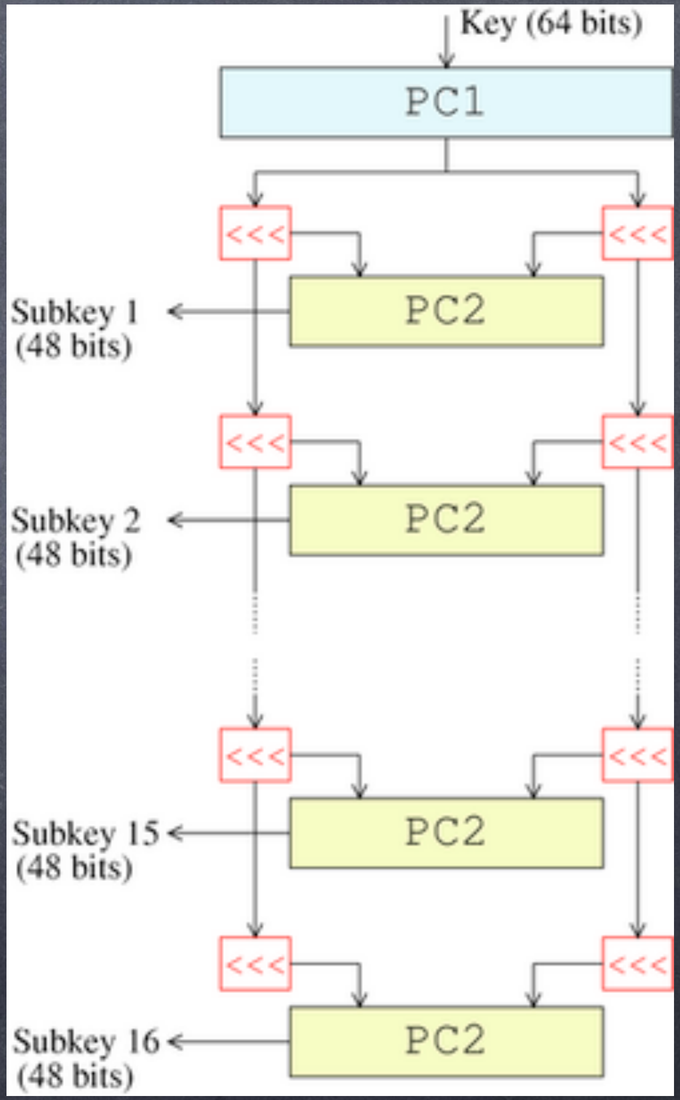
The key schedule of **DES** is relatively simple, with each sub-key $\boldsymbol{k_i}$ being a permuted subset of **48** bits from the master key.

We will not describe the key schedule exactly.

It suffices for us to note that the **56** bits of the master key are divided into two halves -- a "left half " and a "right half " -- each containing **28** bits.

The Design of DES

- In each round, the left-most **24** bits of the sub-key are taken as some subset of the **28** bits in the left half of the master key, and the right-most **24** bits of the sub-key are taken as some subset of the **28** bits in the right half of the master key.



The DES mangler function f′

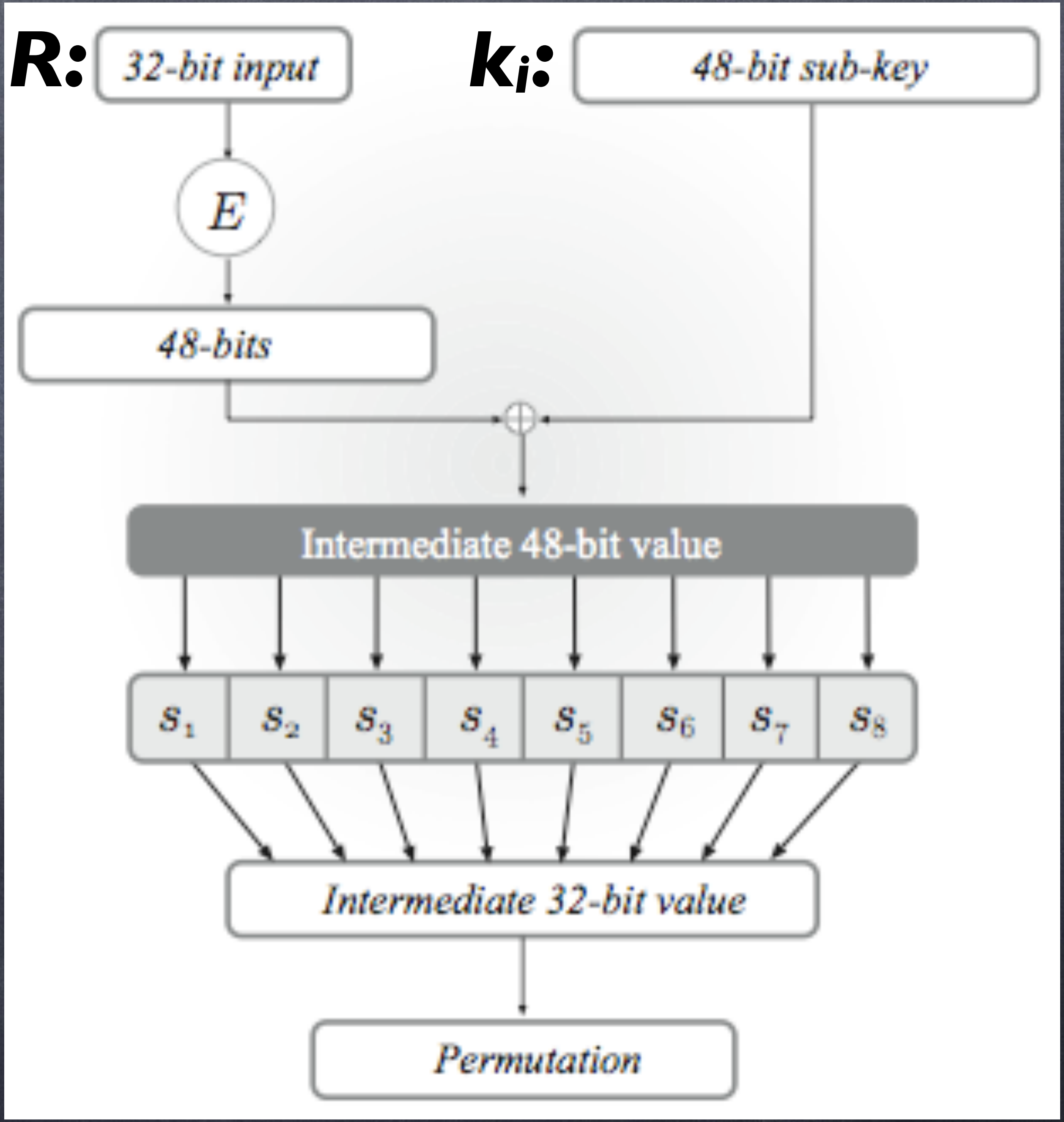
The mangler function in **DES** is constructed using a paradigm we have previously analyzed:

It is (essentially) just a **1**-round substitution-permutation network!

The DES mangler function f'

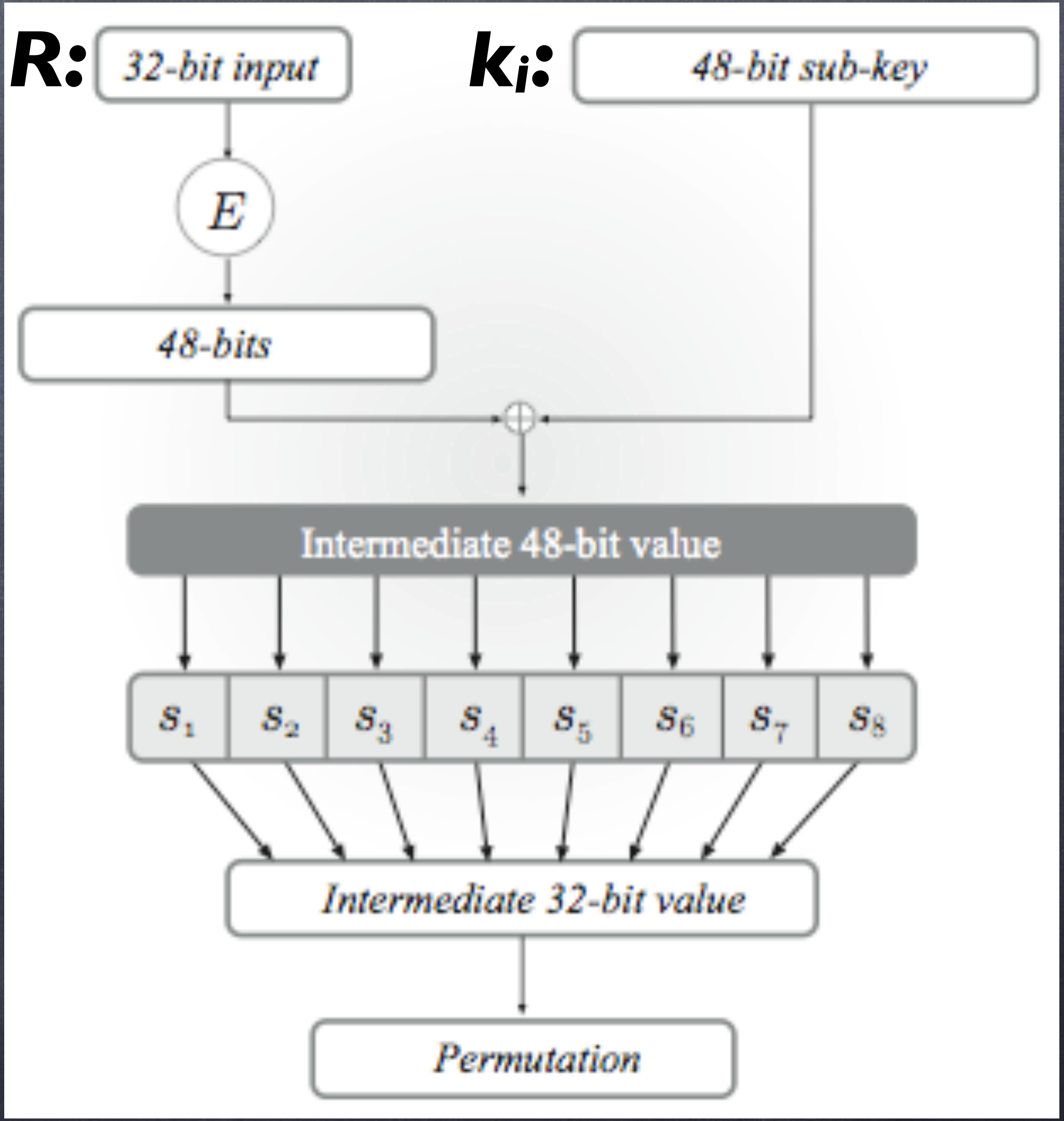
In more detail, computation of $f'(k_i, R)$ with $k_i \in \{0,1\}^{48}$ and $R \in \{0,1\}^{32}$ proceeds as follows:

- First, R is *expanded* to a **48**-bit value R'. This is done by simply duplicating half the bits of R; we denote this by $R' := E(R)$ where E represents the *expansion function.*



The DES mangler function f'

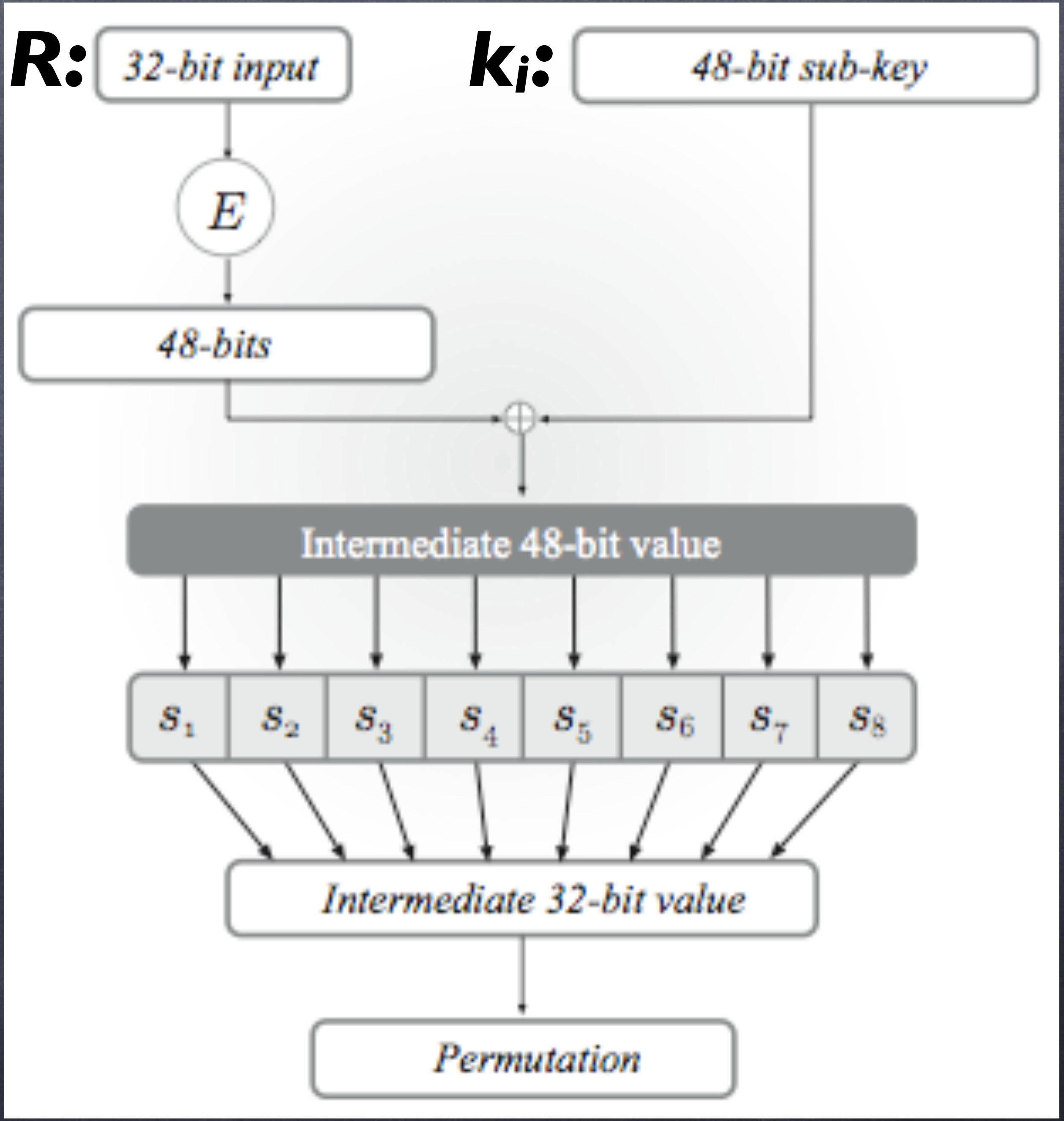
- Following this step, the expanded value ***R'*** is **XOR**ed with ***k_i*** , and the resulting value is divided into **8** blocks, each of which is **6** bits long.



The DES mangler function f′

- Each block is passed through a (different) ***S***-box that takes a **6**-bit input and yields a **4**-bit output; concatenating the output from the **8** ***S***-boxes gives a **32**-bit result.

- As the final step, a mixing permutation is applied to the bits of this result to obtain the final output of ***f***′.



R:
32-bit input
k_i:
48-bit sub-key
E
48-bits
Intermediate 48-bit value
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8
Intermediate 32-bit value
Permutation

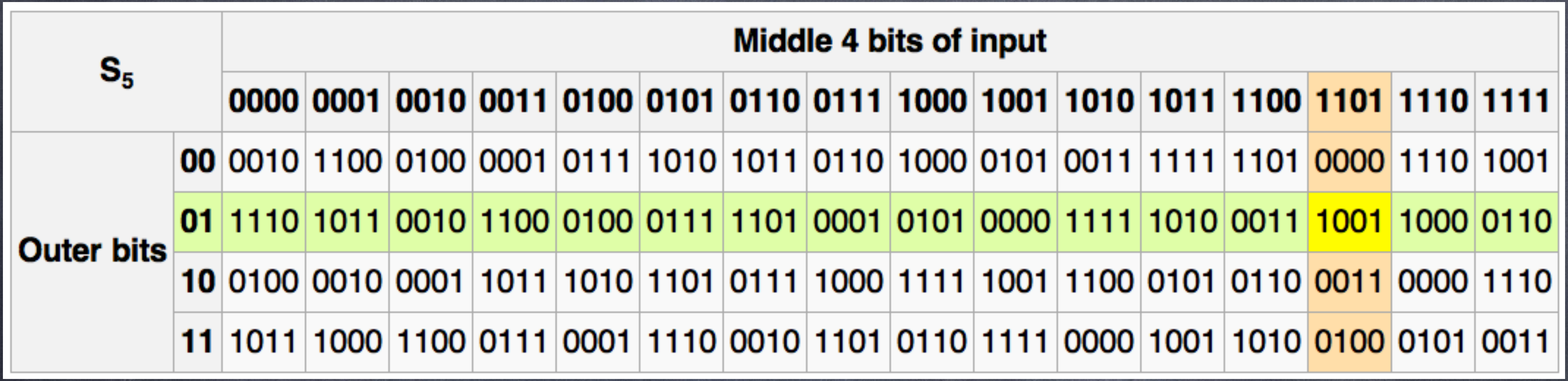
The S-boxes

The eight ***S***-boxes that form the "core" of f' are a crucial element of the **DES** construction, and were very carefully designed (reportedly, with the help of the National Security Agency).

Studies of **DES** have shown that if small changes to the ***S***-boxes had been introduced, or if the ***S***-boxes had been chosen at random, **DES** would have been much more vulnerable to attack.

Warning: seemingly arbitrary choices are not arbitrary at all, and if not made correctly may render the entire construction insecure.

The S-boxes



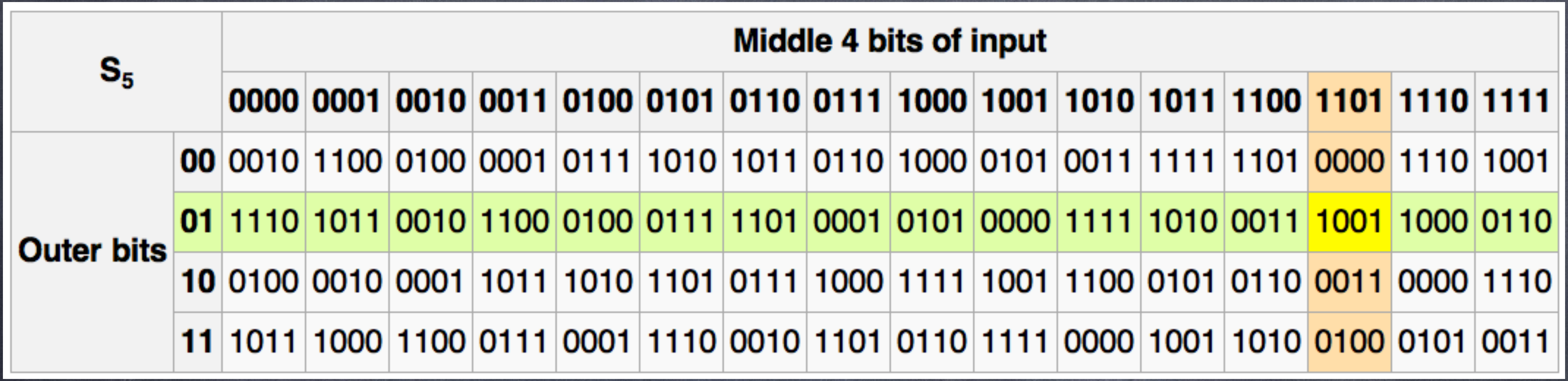
S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Recall that each ***S***-box maps **6**-bit strings to **4**-bit strings.

Each ***S***-box can be viewed as a table with **4** rows and **16** columns, where each cell of the table contains a 4-bit entry.

A **6**-bit input can be viewed as indexing one of the **2^6 = 64** cells of the table in the following way:

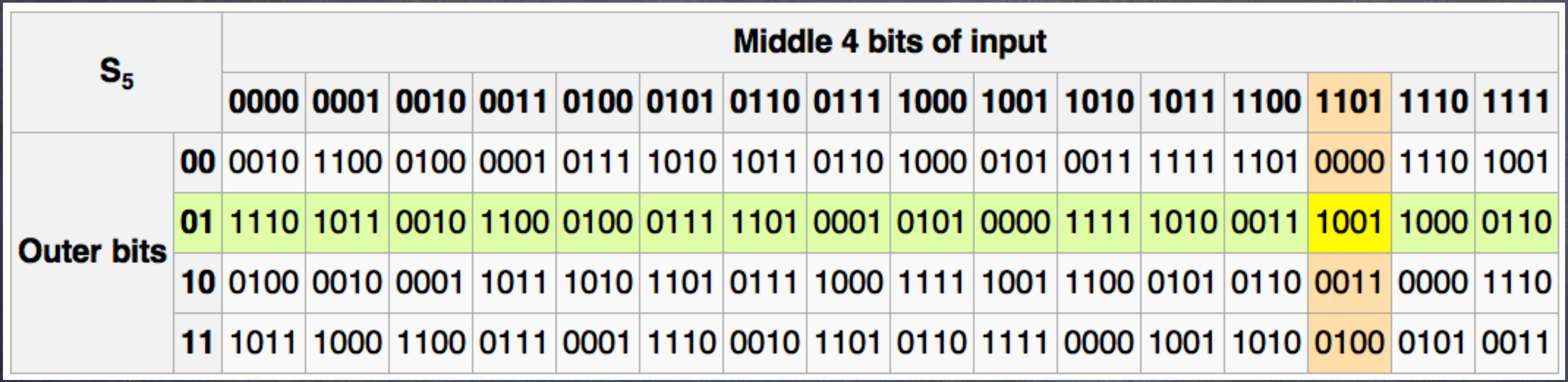
The S-boxes



S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

- The first and last input bits are used to choose the table row, and bits **2–5** are used to choose the column.
- The **4**-bit entry at a particular cell represents the output value for the input associated with that position.

The S-boxes



S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

1. Each ***S***-box is a **4**-to-**1** function.

2. Each row in the table contains each of the **16** possible **4**-bit strings exactly once. (That is, each row is a *permutation* of the **16** possible 4-bit strings.)

3. Changing *one bit* of the input always changes at least *two bits* of the output.

The DES avalanche effect

The third property of the **DES** ***S***-boxes described above, along with the mixing permutation that is used in the mangler function, ensure that **DES** exhibits a strong avalanche effect.

The DES avalanche effect

DES has **16** rounds, and so the avalanche effect is completed early (**8**th round) in the computation.

This ensures that the computation of **DES** on similar inputs yields completely independent-looking outputs.

We remark that the avalanche effect in **DES** is also due to a careful choice of the mixing permutation, and in fact it has been shown that a *random* mixing permutation would yield weaker effect.

INTRODUCTION TO
MODERN
CRYPTOGRAPHY
Second Edition
Jonathan Katz •Yehuda Lindell

Chapter 6 : Practical Constructions of Symmetric-Key Primitives

Attacks on Reduced-Round Variants

Clearly **DES** with three rounds or fewer cannot be a pseudorandom function because the avalanche effect is not yet complete after only three rounds.

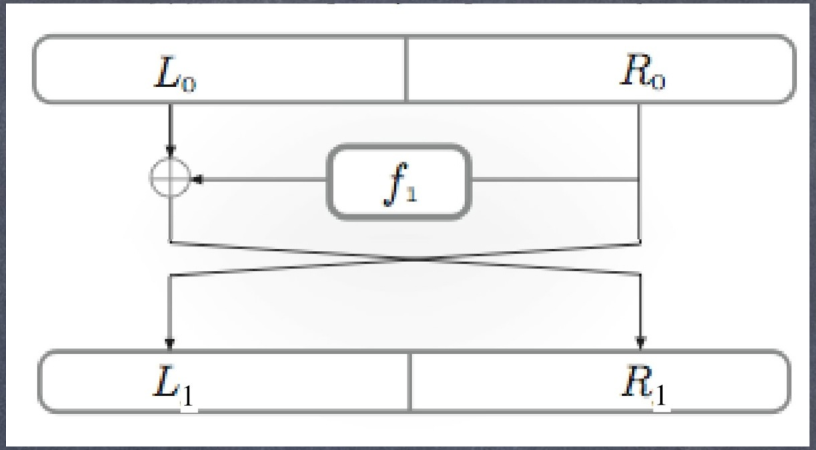
Thus, we will be interested in demonstrating more difficult (and more damaging) *key-recovery attacks* which compute the key ***k*** using only a relatively small number of input/output pairs computed using that key.

Attacks on Reduced-Round Variants of DES

All the attacks below will be known-plaintext attacks whereby the adversary has plaintext/ciphertext pairs $\{(x_i,y_i)\}$ with $y_i = DES_k(x_i)$ for some secret key k.

When we describe the attacks, we will focus on a particular input/output pair (x, y) and will describe the information about the key that an adversary can derive from this pair.

Single-round DES



In single-round **DES**, we have that $\boldsymbol{y = (L_1, R_1)}$ where

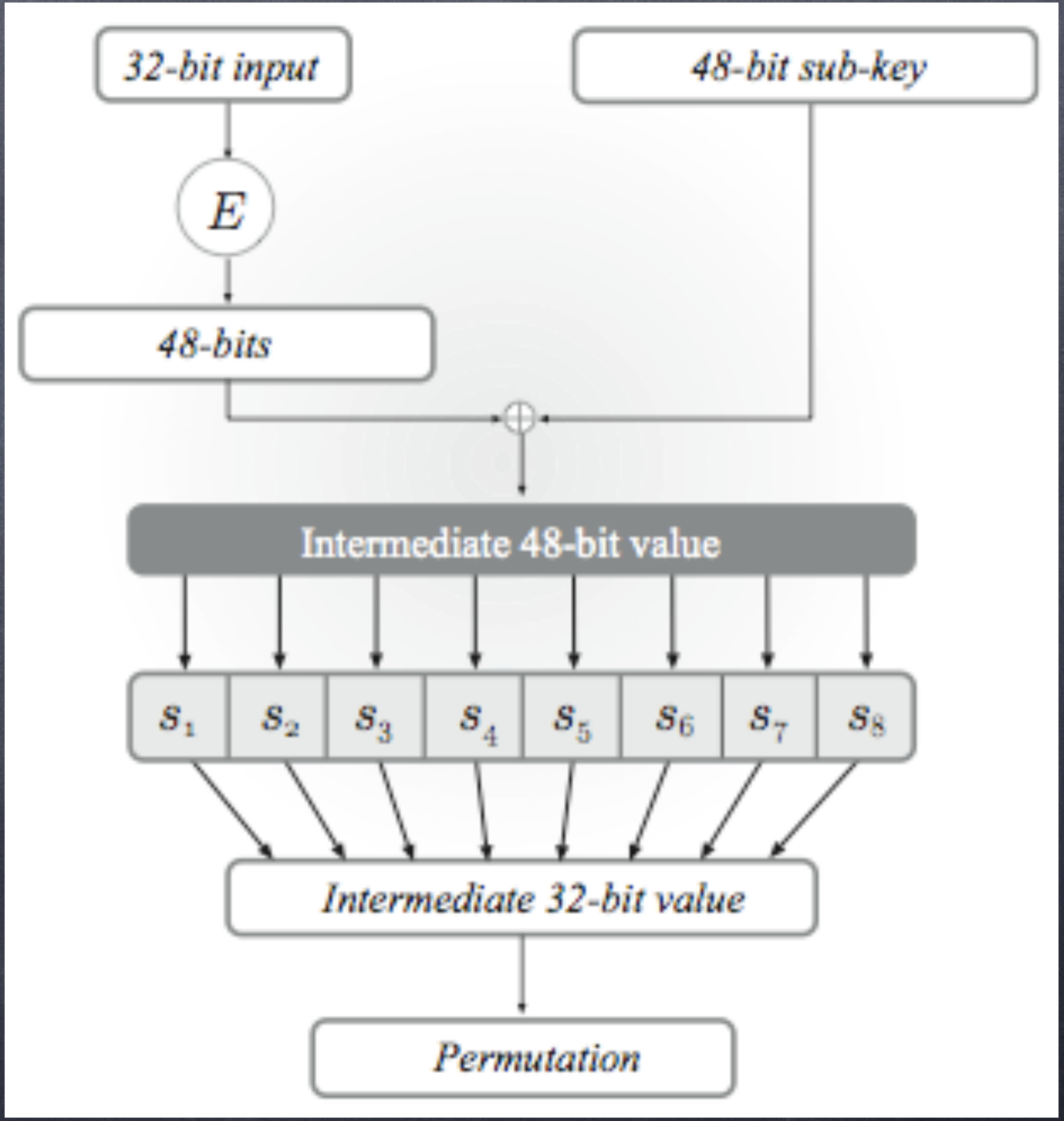
$$L_1 = R_0 \text{ and } R_1 = L_0 \oplus f_1(R_0).$$

We therefore know an input/output pair for f_1; specifically, we know that $f_1(R_0) = R_1 \oplus L_0$ (all these values are known).

Single-round DES

By applying the inverse of the mixing permutation to the output $R_1 \oplus L_0$, we obtain the intermediate value that contains the outputs from all the ***S***-boxes, where the first **4** bits are the output from the first ***S***-box, the next **4** bits are the output from the second ***S***-box, and so on.

This means that we have the exact output of each ***S***-box.


32-bit input
48-bit sub-key
E
48-bits
Intermediate 48-bit value
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
Intermediate 32-bit value
Permutation

Single-round DES

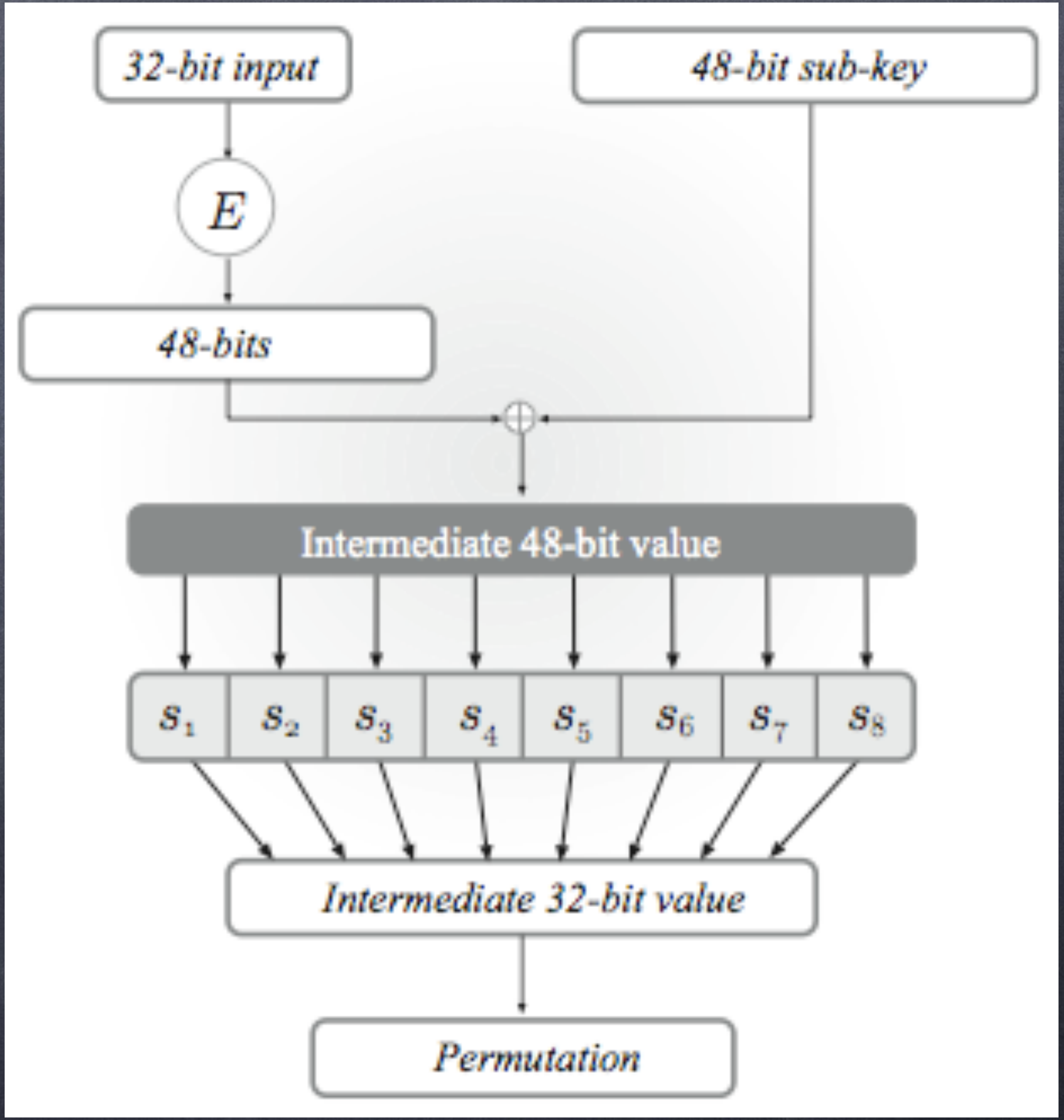
Consider the (known) **4**-bit output of the first ***S***-box.

Recalling that each ***S***-box is a **4**-to-**1** function, this means that there are exactly four possible inputs to this ***S***-box that would result in the given output, and similarly for all the other ***S***-boxes; each such input is **6** bits long.

Single-round DES

The input to the ***S***-boxes is simply the **XOR** of ***E*(*R*$_0$)** with the key ***k*$_1$** used in this round. (Actually, for single-round **DES**, ***k*$_1$** is the only key.)

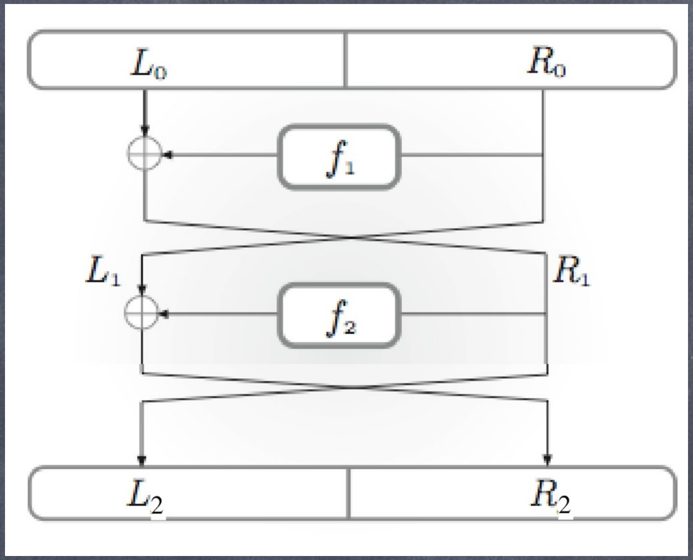
Since ***R*$_0$** is known, we conclude that for each **6**-bit portion of ***k*$_1$** there are four possible values (and compute them).



Single-round DES

- This means we have reduced the number of possible keys $\boldsymbol{k_1}$ from $\mathbf{2^{48}}$ to $\mathbf{4^{48/6} = 4^8 = 2^{16}}$ (since there are four possibilities for each of the eight **6**-bit portions of $\boldsymbol{k_1}$).
- This is already a small number and so we can just try all the possibilities on a different input/output pair $\mathbf{(x', y')}$ to find the right one.
- Recover full key using two known plaintexts in time $\mathbf{2^{16}}$.

Two-round DES



In two-round **DES**, the output **y** is equal to (L_2, R_2) where

$$L_1 = R_0 \qquad\qquad R_1 = L_2 = L_0 \oplus f_1(R_0)$$

$$L_2 = R_1 = L_0 \oplus f_1(R_0) \qquad R_2 = L_1 \oplus f_2(R_1).$$

Note that L_0, R_0, L_2, R_2 are known from the input/output pair and thus we also know

$$L_0 \oplus L_2 = f_1(R_0) \quad \text{and} \quad R_0 \oplus R_2 = f_2(L_2).$$

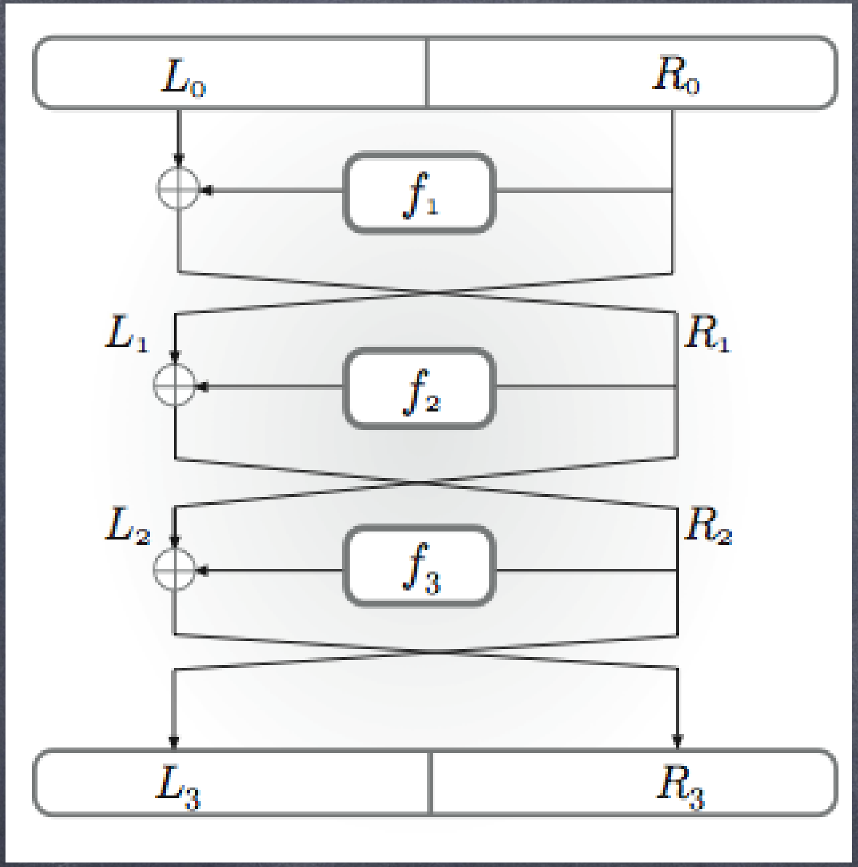
Two-round DES

This means that we know the input/output of f_1 and f_2 , and so the same method used in the attack on single-round **DES** can be used here to determine both k_1 and k_2 in time roughly $\mathbf{2 \cdot 2^{16}}$.

This attack works even if k_1 and k_2 are completely independent keys.

In fact, the key schedule of **DES** ensures that many of the bits of k_1 and k_2 are equal, which can be used to further speed up the attack.

Three-round DES



The time complexity of the attack is roughly $2 \cdot 2^{28} + 2^{24} < 2^{30}$, and its space complexity is $2 \cdot 2^{12}$. An attack of this complexity could be carried out on a standard personal computer.

The Security of DES

After almost **30** years of intensive study, the best known practical attack on **DES** is still just an exhaustive search through its key space.

Unfortunately, the **56**-bit key length of **DES** is short enough that an exhaustive search through all $\mathbf{2^{56}}$ possible keys is now feasible (though still non-trivial).

The Security of DES

- Already in the late ’**70**s there were strong objections to the choice of such a short key for **DES**.

- Back then, the objection was theoretical as the computational power needed to search through that many keys was generally unavailable.

- The practicality of a brute force attack on **DES** nowadays, however, was demonstrated in **1997** when a number of **DES** challenges set up by **RSA** Security were solved.

The Security of DES

The first challenge was broken in **1997** by the **DESCHALL** project using thousands of computers coordinated across the Internet; the computation took **96** days.

A second challenge was broken the following year in just **41** days by the distributed.net project.

- A significant breakthrough came later in **1998** when the third challenge was solved in just ***56** hours*.

The Security of DES

- This impressive feat was achieved via a special-purpose **DES**-breaking machine called *Deep Crack* that was built by the Electronic Frontier Foundation at a cost of **$250,000**.

- The latest challenge was solved in just over **22** hours (a combined effort of Deep Crack and distributed.net).

- The bottom line is that **DES** has a key that is far too short and cannot be considered secure for any application today.

The Security of DES

Looking ahead a bit, we note that the Advanced Encryption Standard (**AES**) — the replacement for **DES** — was explicitly designed to address concerns regarding the short key length and block length of **DES**.

AES supports keys of length **128** bits (and more), and a block length of **128** bits.

6.2.6 Advanced Cryptanalytic Attacks on DES

- The successful brute-force attacks described above do not utilize any internal weaknesses of **DES**.
- Indeed, for many years no such weaknesses were known.
- The first breakthrough on this front was by Biham and Shamir in the late ’**80**s who developed a technique called *differential cryptanalysis* and used it to design an attack on **DES** using less time than a brute-force search.

Advanced Cryptanalytic Attacks on DES

- Their specific attack takes time $\mathbf{2^{37}}$ (and uses negligible memory) but requires the attacker to analyze $\mathbf{2^{36}}$ ciphertexts obtained from a pool of $\mathbf{2^{47}}$ chosen plaintexts.

- While the existence of this attack was a breakthrough result from a theoretical standpoint, it does not appear to be of much practical concern since it is hard to imagine any realistic scenario where an adversary can obtain this many values in a chosen-plaintext attack.

Advanced Cryptanalytic Attacks on DES

Eli Biham and Adi Shamir

Differential Cryptanalysis

- Let $\Delta_x, \Delta_y \in \{0, 1\}^n$. We say that the differential (Δ_x, Δ_y) appears with probability p if for random inputs x_1 and $x_2 = x_1 \oplus \Delta_x$ and random choice of key k, the probability that $F_k(x_1) \oplus F_k(x_2) = \Delta_y$ is p.
- It is clear that for a random function, no differential should appear with probability p much higher than 2^{-n}.

Advanced Cryptanalytic Attacks on DES

1. Interestingly, the work of Biham and Shamir indicated that the **DES** ***S***-boxes had been specifically designed to be resistant to differential cryptanalysis (to some extent), suggesting that the technique of differential cryptanalysis was known (but not publicly revealed) by the designers of **DES**.

Advanced Cryptanalytic Attacks on DES

2. After Biham and Shamir announced their result, the designers of **DES** (represented by Don Coppersmith) claimed that they were indeed aware of differential cryptanalysis and had designed **DES** to thwart this type of attack (but were asked by the **NSA** to keep it quiet in the interests of national security).

Don Coppersmith

6.2.4 Increasing the Key Length of a Block Cipher

- The only known practical weakness of **DES** is its relatively short key.
- It thus makes sense to try to design a block cipher with a larger key length using "basic" **DES** as a building block.
- Some approaches to doing so are discussed in this section.

Internal tampering vs. black-box constructions.

- Although we refer to **DES** throughout the discussion, and **DES** is the most prominent instance where these techniques have been applied, everything we say here applies generically to any block cipher.
- The first approach would be to somehow modify the *internal structure* of **DES**, while increasing the key length.

Internal tampering vs. black-box constructions

- For example, one could leave the mangler function untouched and simply use a **128**-bit master key with a different key schedule (still choosing a 48-bit sub-key in each round).
- Or, one could change the ***S***-boxes themselves and use a larger sub-key in each round.
- The disadvantage of this approach is that by modifying **DES** — in even the smallest way — we lose the confidence we have gained in **DES** by virtue of the fact that it has remained secure for so many years.

Internal tampering vs. black-box constructions

- More to the point, cryptographic constructions are very sensitive and thus even mild, seemingly-insignificant changes can render the original construction completely insecure.
- Changing the internals of a block cipher is therefore not recommended.

Internal tampering vs. black-box constructions

- An alternative approach that does not suffer from the above problem is to use **DES** as a "black box".
- That is, in this approach we completely ignore the internal structure of **DES** and treat it as a black box that implements a "perfect" block cipher with a **56**-bit key.

Internal tampering vs. black-box constructions

- Then, a new cipher is constructed that uses only invocations of the original unmodified **DES**.
- Since **DES** itself is not changed at all, this approach is much more likely to lead to a secure cipher (though it may also lead to a less efficient one).

Double Encryption

Let F be a block cipher. Then a new block cipher F' with a key that is twice the length of the original one is defined by

$$F'_{k_1,k_2}(x) \overset{\text{def}}{=} F_{k_2}(F_{k_1}(x)),$$

where k_1 and k_2 are independent keys.

If F is **DES** then F' is a block cipher taking a **112**-bit key.

If exhaustive search were the best available attack on F', a key length of **112** bits would be sufficient.

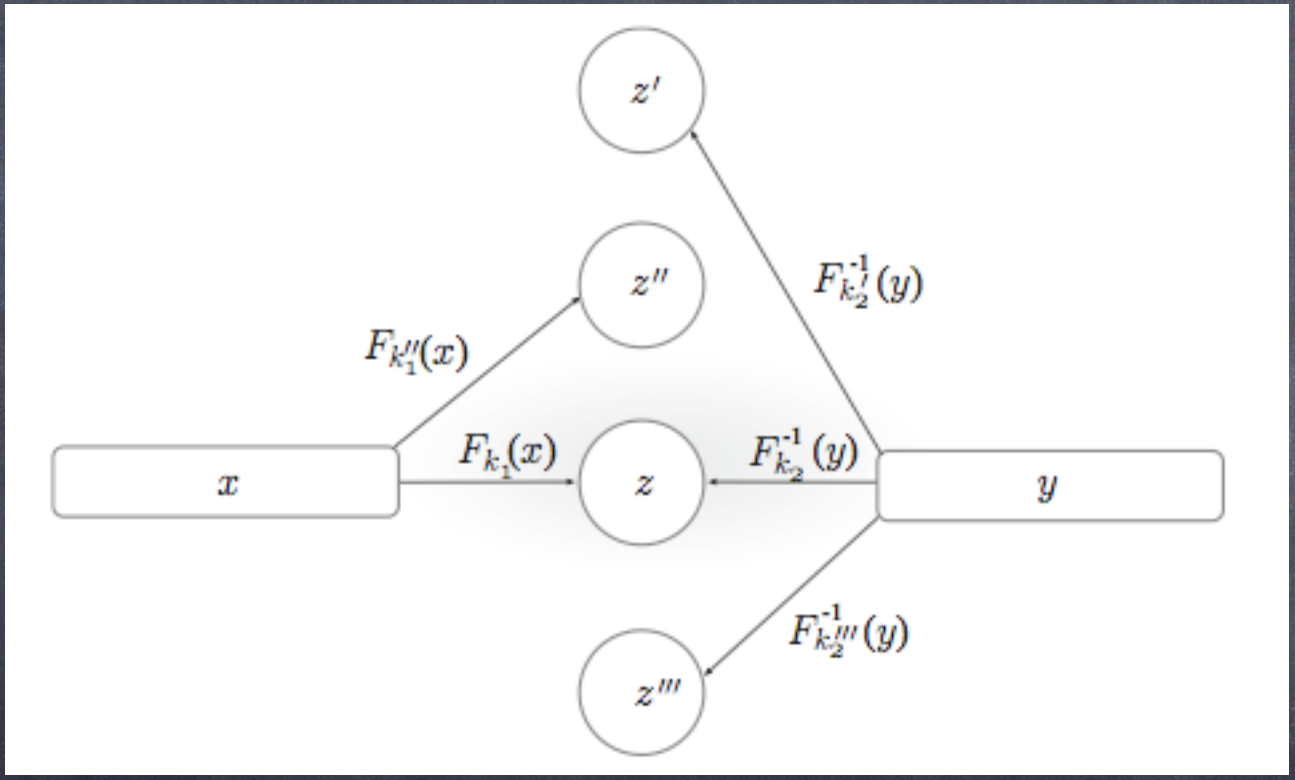
Double Encryption

Unfortunately, we now show an attack on F' that runs in time roughly 2^n when the original keys k_1 and k_2 are each of length n (and the block length is at least n);

This is significantly less than the 2^{2n} time one would hope would be necessary to carry out an exhaustive search for a $2n$-bit key.

This means that the new block cipher is essentially no better than the old one, even though it has a key that is twice as long.

meet-in-the-middle attack



meet-in-the-middle attack

Say the adversary is given a single input/output pair **(x, y)** where **$y = F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$**. The adversary will narrow down the set of possible keys in the following way:

1. Set $S := \emptyset$.

2. For each $k_1 \in \{0,1\}^n$, compute $z := F_{k_1}(x)$ and store (z, k_1) in a list L.

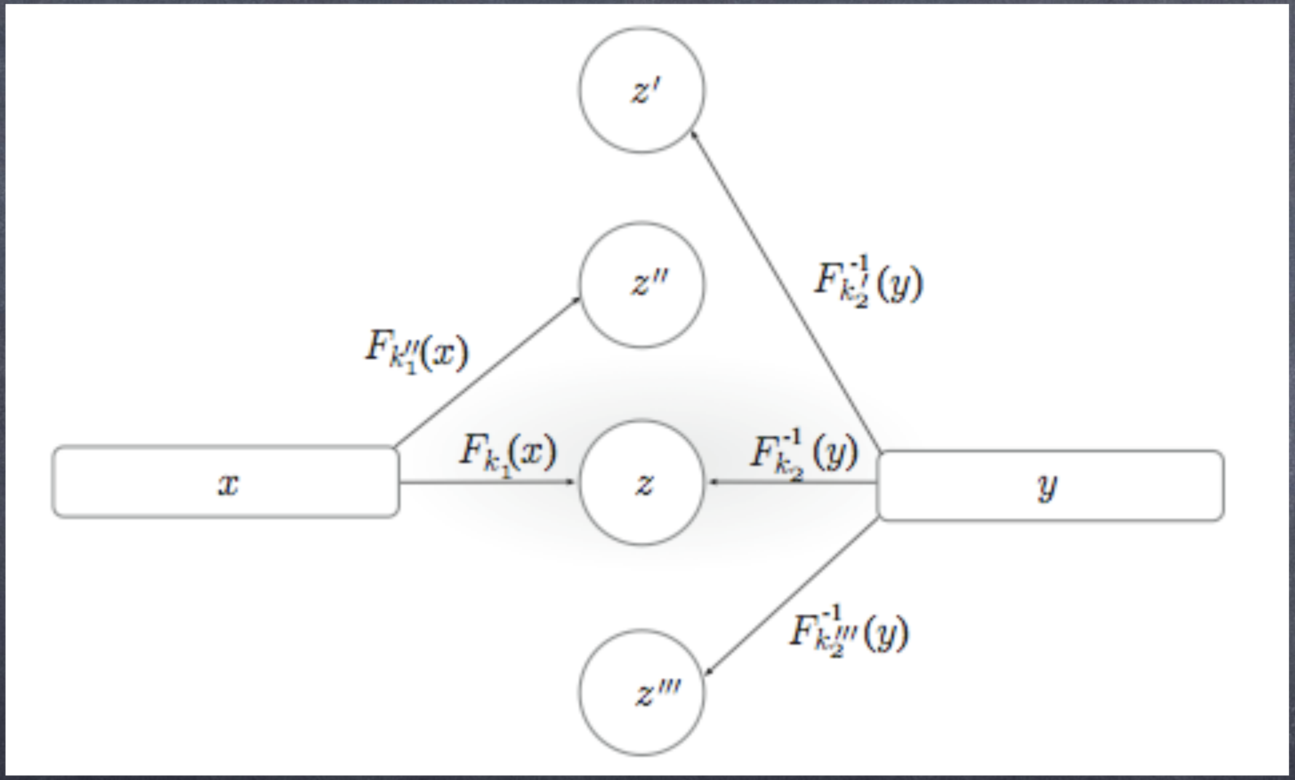
3. For each $k_2 \in \{0,1\}^n$, compute $z := F^{-1}_{k_2}(y)$ and store (z, k_2) in a list L'.

meet-in-the-middle attack

4. Sort $\boldsymbol{L}$ and $\boldsymbol{L'}$, respectively, by their first components.

5. Say that an entry $(\boldsymbol{z_1},\boldsymbol{k_1})$ in $\boldsymbol{L}$ and another entry $(\boldsymbol{z_2},\boldsymbol{k_2})$ in $\boldsymbol{L'}$ are a *match* if $\boldsymbol{z_1} = \boldsymbol{z_2}$. For each match of this sort, add $(\boldsymbol{k_1},\boldsymbol{k_2})$ to $\boldsymbol{S}$.

meet-in-the-middle attack



meet-in-the-middle attack

The set $\boldsymbol{S}$ output by this algorithm contains exactly those values $(\boldsymbol{k_1},\boldsymbol{k_2})$ for which $\boldsymbol{y = F'_{k_1,k_2}(x)}$.

This holds because it outputs exactly those values $(\boldsymbol{k_1},\boldsymbol{k_2})$ satisfying

$$\boldsymbol{F_{k_1}(x) = F^{-1}_{k_2}(y)},$$

which holds if and only if $\boldsymbol{y = F'_{k_1,k_2}(x)}$.

meet-in-the-middle attack

If n is also the block length of F then a random pair (k_1, k_2) is expected to satisfy Equation (5.3) with probability roughly 2^{-n} , and so the number of elements in S will be approximately $2^{2n} / 2^n = 2^n$.

Given another two input/output pairs and trying all 2^n elements of S with respect to these pairs is expected to identify the correct (k_1, k_2) with very high probability.

Triple Encryption

The obvious generalization of the preceding approach is to apply the block cipher three times in succession.

Two variants of this approach are common:

1. *Variant 1 — three independent keys:*
Choose 3 independent keys $\boldsymbol{k_1, k_2, k_3}$ and define

$$F'_{k_1,k_2,k_3}(x) \overset{\text{def}}{=} F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x))).$$

Triple Encryption

The obvious generalization of the preceding approach is to apply the block cipher three times in succession.

Two variants of this approach are common:

2. *Variant 2 — two independent keys:*
Choose 2 independent keys $\boldsymbol{k_1}, \boldsymbol{k_2}$ and define

$$F'_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x))).$$

Triple Encryption

Before comparing the security of the two alternatives we note that the middle invocation of the original cipher is actually in the reverse direction.

If ***F*** is a sufficiently good cipher this makes no difference to the security, since if ***F*** is a strong pseudorandom permutation then F^{-1} must be too.

Triple Encryption

- The reason for this strange alternation between $\boldsymbol{F}$, $\boldsymbol{F^{-1}}$, and $\boldsymbol{F}$ is so that if one chooses $\boldsymbol{k_1=k_2=k_3}$,the result is a single invocation of $\boldsymbol{F}$ with $\boldsymbol{k_1}$.
- This ensures backward compatibility (i.e., in order to switch back to a single invocation of $\boldsymbol{F}$, it suffices to just set the keys to all be equal).

Security of the <u>first</u> variant

The key length of this variant is $\mathbf{3}\boldsymbol{n}$ (where, as before, the key length of the original cipher $\boldsymbol{F}$ is $\boldsymbol{n}$) and so we might hope that the best attack on this cipher would require time $\mathbf{2}^{\mathbf{3}\boldsymbol{n}}$.

However, the cipher is susceptible to a meet-in-the-middle attack just as in the case of double encryption, though the attack now takes time $\mathbf{2}^{\mathbf{2}\boldsymbol{n}}$.

This is the best known attack.

Thus, although this variant is not as secure as we might have hoped, it obtains sufficient security for all practical purposes if $\boldsymbol{n} = \mathbf{56}$.

Security of the <u>second</u> variant

The key length of this variant is $\mathbf{2}\boldsymbol{n}$ and so the best we can hope for is security against attacks running in time $\mathbf{2}^{\mathbf{2}\boldsymbol{n}}$.

There is no known attack with better time complexity when the adversary is given only a single input/output pair.

However, there is a known chosen-plaintext attack that finds the key in time $\mathbf{2}^{\boldsymbol{n}}$ using $\mathbf{2}^{\boldsymbol{n}}$ <u>chosen input/output pairs</u>. Despite this, it is still a reasonable choice in practice.

Triple-DES (3DES)

Triple-**DES** is based on a triple invocation of DES using two or three keys, as described above.

It is widely believed to be highly secure and in **1999** officially replaced **DES** as a standard.

Triple-**DES** is still widely used today and is considered a very strong block cipher.

Its only drawbacks are its relatively small block length and the fact that it is quite slow since it requires **3** full block cipher operations.

COMP547 Claude Crépeau

INTRODUCTION TO
MODERN
CRYPTOGRAPHY
Second Edition
Jonathan Katz •Yehuda Lindell

Chapter 6 : Practical Constructions of Symmetric-Key Primitives

6.2.5 AES – The Advanced Encryption

In January **1997**, the United States National Institute of Standards and Technology (**NIST**) announced that it would hold a competition to select a new block cipher — to be called the *Advanced Encryption Standard*, or **AES** — to replace **DES**.

The Advanced Encryption Standard

The competition began with an open call for teams to submit candidate block ciphers for evaluation.

A total of **15** different algorithms were submitted from all over the world, and these submissions included the work of many of the very best cryptographers and cryptanalysts today.

Each team's candidate cipher was intensively analyzed by **NIST**, the public, the other teams.

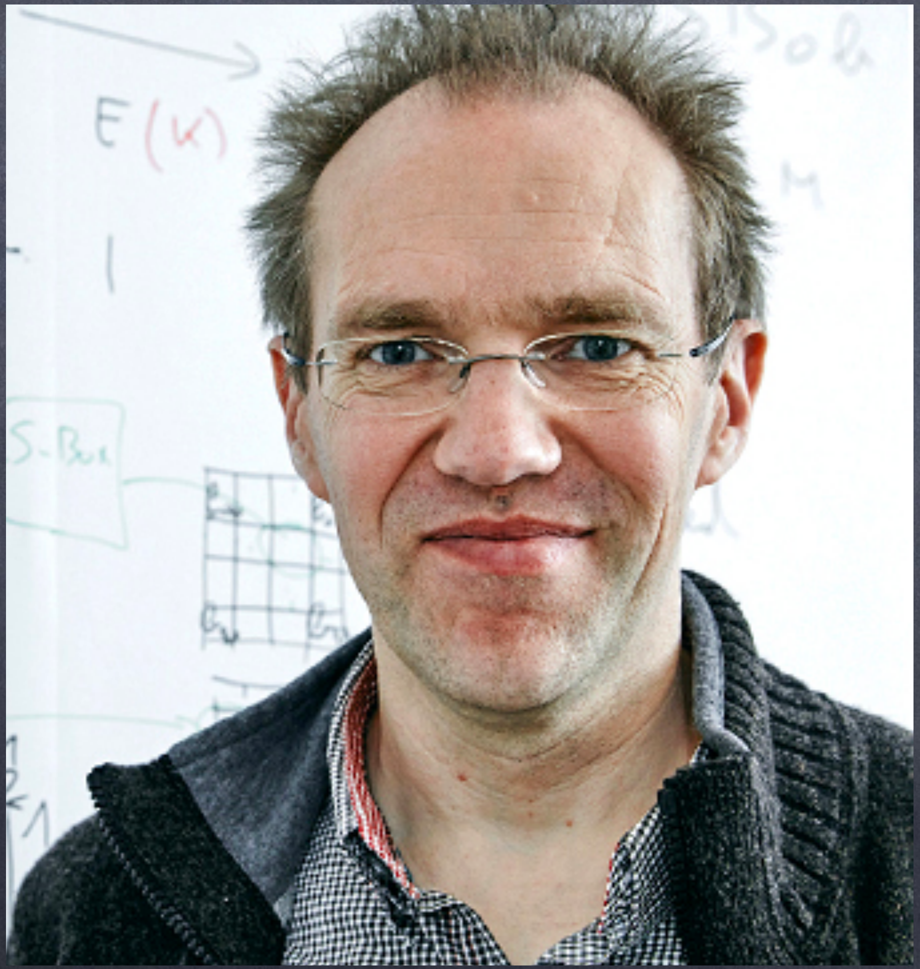
The Advanced Encryption Standard

Two workshops were held, one in **1998** and one in **1999**, at which cryptanalytic attacks of the various submissions were shown.

Following the second workshop, **NIST** narrowed the field down to **5** "finalists" and the second round of the competition began.

A third **AES** workshop was held, inviting additional scrutiny on the five finalists.

The Advanced Encryption Standard



Joan Daemen and Vincent Rijmen

The Advanced Encryption Standard

In October **2000**, **NIST** announced that the winning algorithm was **Rijndael** (a block cipher designed by Joan Daemen and Vincent Rijmen from Belgium), though it conceded that any of the **5** finalists would have made an excellent choice.

In particular, no serious security vulnerabilities were found in any of the **5** finalists, and the selection of a “winner” was based in part on properties such as efficiency, flexibility, etc.

The AES construction

The **AES** block cipher has a **128**-bit block length and can use **128**-bit, **192**-bit, or **256**-bit keys.

The length of the key affects the key schedule (i.e., the sub-key that is used in each round) as well as the number of rounds, but does not affect the high-level structure of each round.

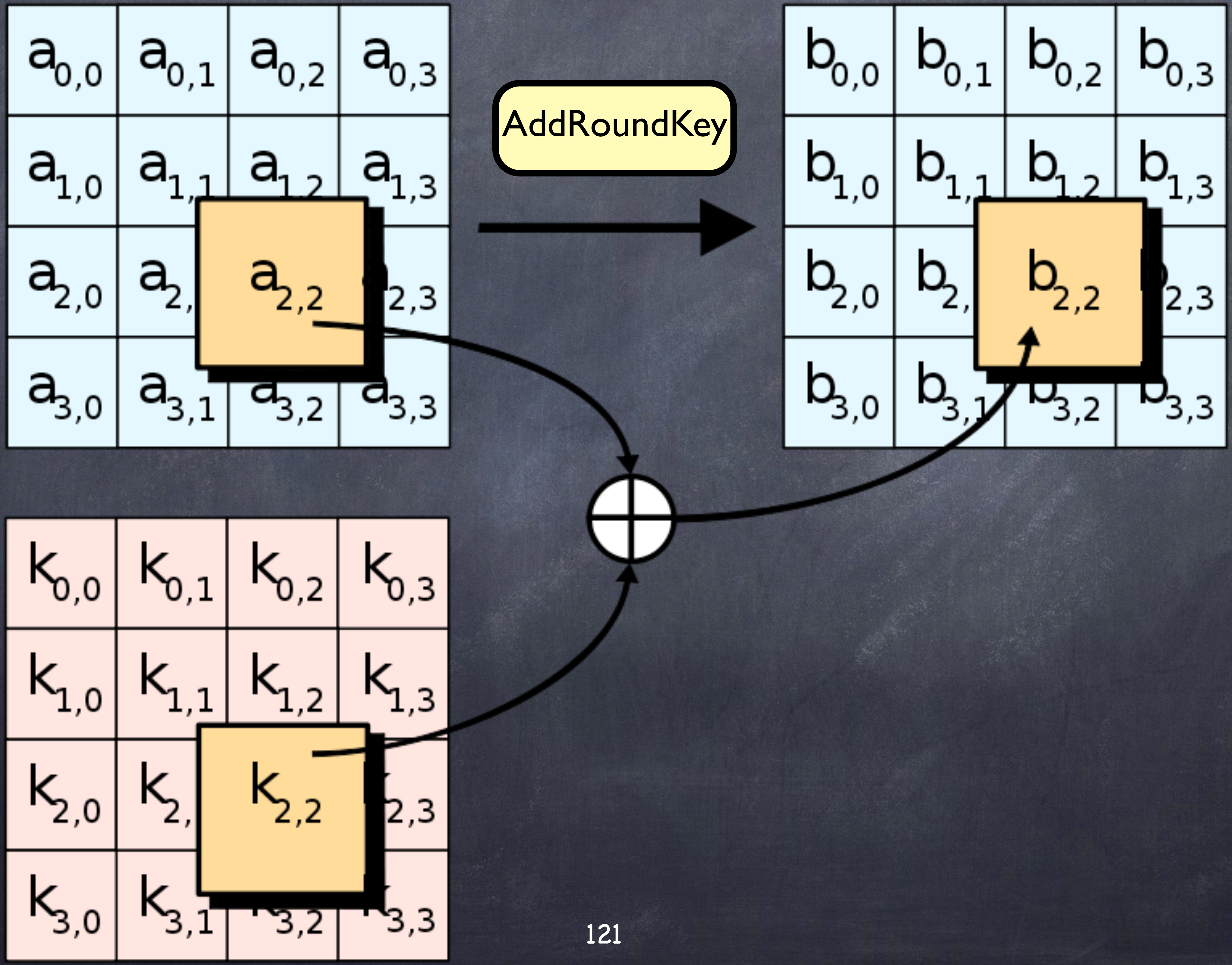
The AES construction

In contrast to **DES** that uses a Feistel structure, **AES** is essentially a substitution-permutation network.

During computation of the **AES** algorithm, a **4**-by-**4** array of bytes called the *state* is modified in a series of rounds.

The state is initially set equal to the input to the cipher (note that the input is **128** bits which is exactly **16** bytes).

The AES construction



AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$\oplus$

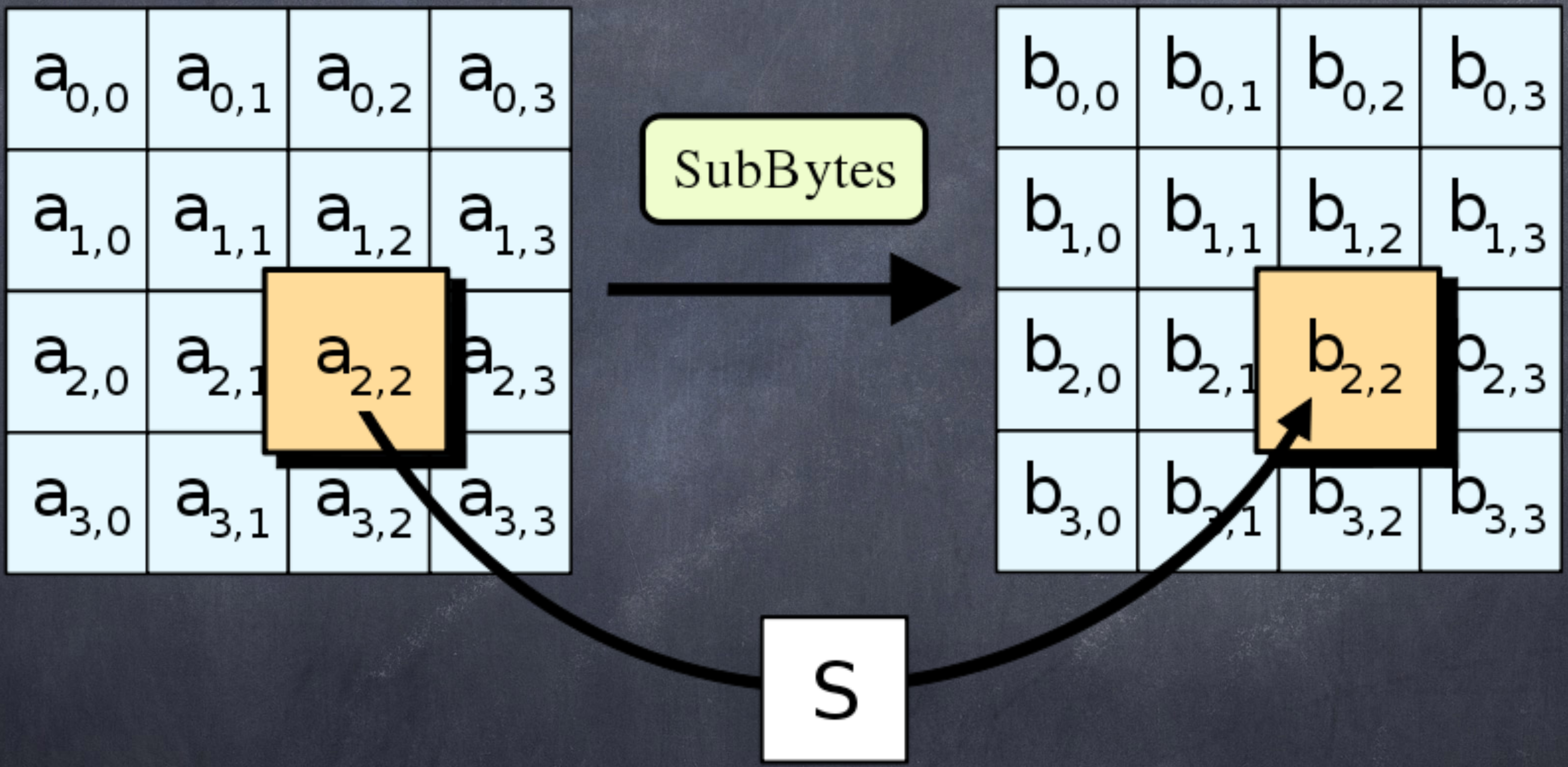
The AES construction

Stage 1 — **AddRoundKey**:

In every round of **AES**, a **128**-bit sub-key is derived from the master key, and is interpreted as a **4**-by-**4** array of bytes.

The state array is updated by **XOR**ing it with this sub-key.

The AES construction


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

The AES construction

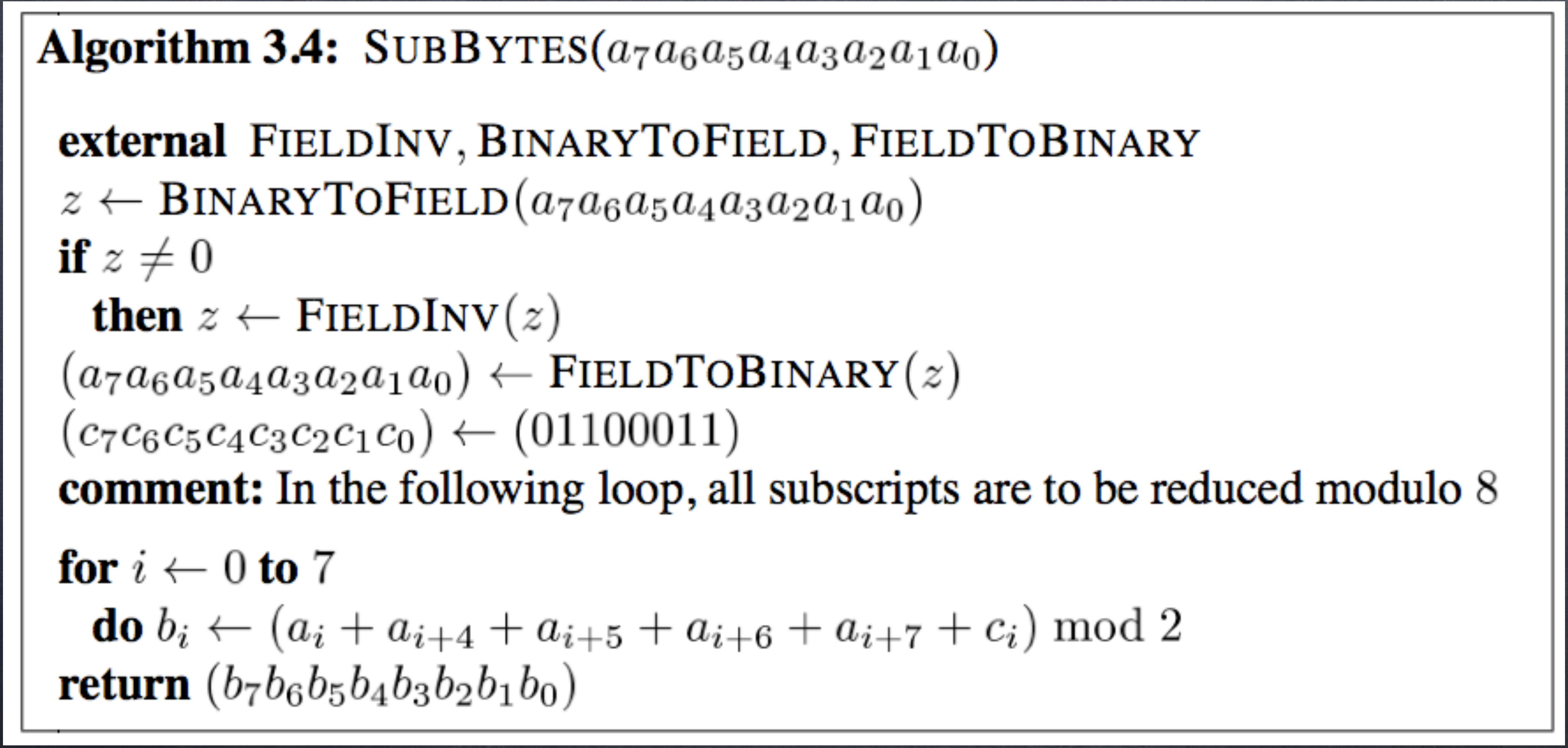
Stage 2 — **SubBytes**:

In this step, each byte of the state array is replaced by another byte according to a single fixed lookup table ***S*** .

This ***S***-box is a bijection over $\{\mathbf{0},\mathbf{1}\}^{\mathbf{8}}$.
We stress that there is only *one* ***S***-box and it is used for substituting *all* the bytes in the state array, in every round.

$$x^{-1*} = \begin{cases} x^{-1} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

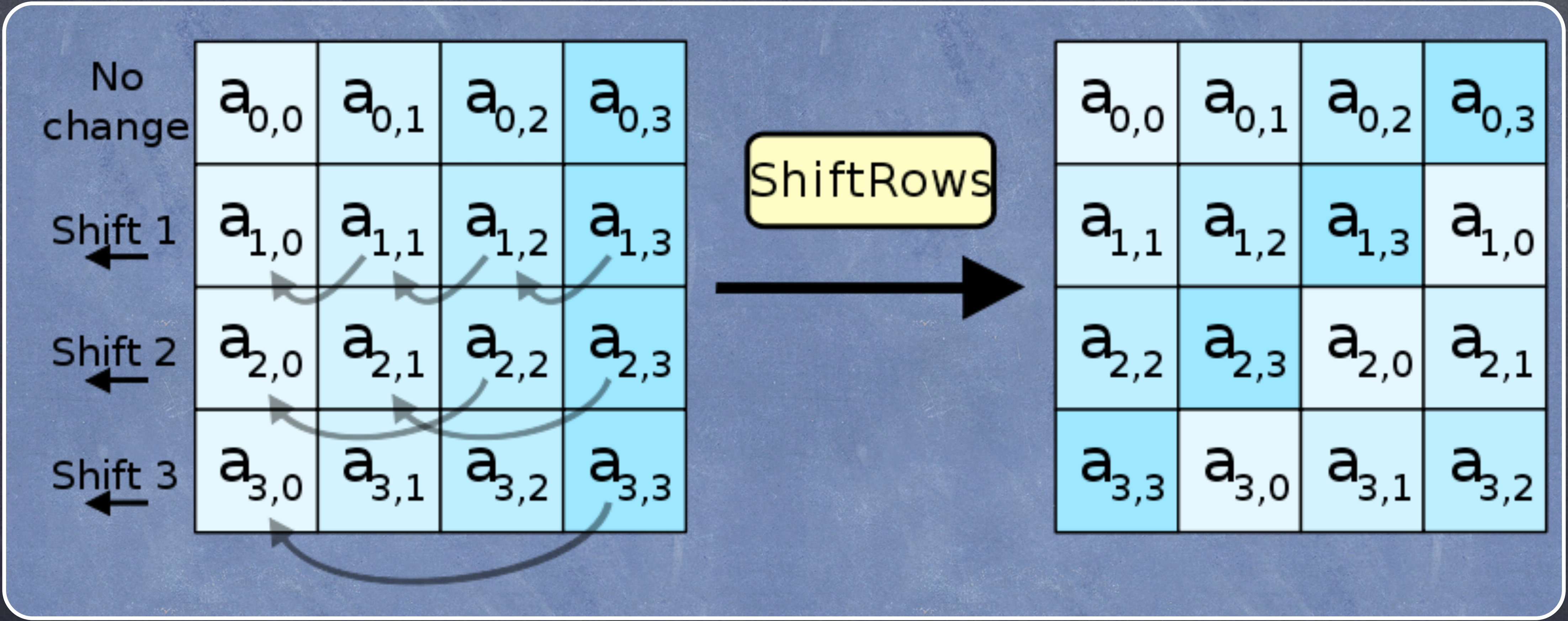
$$\begin{pmatrix} 1&1&1&1&0&0&0&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 0&0&0&1&1&1&1&1 \\ 0&0&1&1&1&1&1&0 \\ 0&1&1&1&1&1&0&0 \\ 1&1&1&1&1&0&0&0 \end{pmatrix} \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}^{-1*} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix}$$



```
**Algorithm 3.4:** SUBBYTES($a_7a_6a_5a_4a_3a_2a_1a_0$)

**external** FIELDINV, BINARYTOFIELD, FIELDTOBINARY
$z \leftarrow$ BINARYTOFIELD($a_7a_6a_5a_4a_3a_2a_1a_0$)
**if** $z \neq 0$
  **then** $z \leftarrow$ FIELDINV($z$)
$(a_7a_6a_5a_4a_3a_2a_1a_0) \leftarrow$ FIELDTOBINARY($z$)
$(c_7c_6c_5c_4c_3c_2c_1c_0) \leftarrow (01100011)$
**comment:** In the following loop, all subscripts are to be reduced modulo 8
**for** $i \leftarrow 0$ **to** $7$
  **do** $b_i \leftarrow (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \bmod 2$
**return** $(b_7b_6b_5b_4b_3b_2b_1b_0)$
```

The AES construction


No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

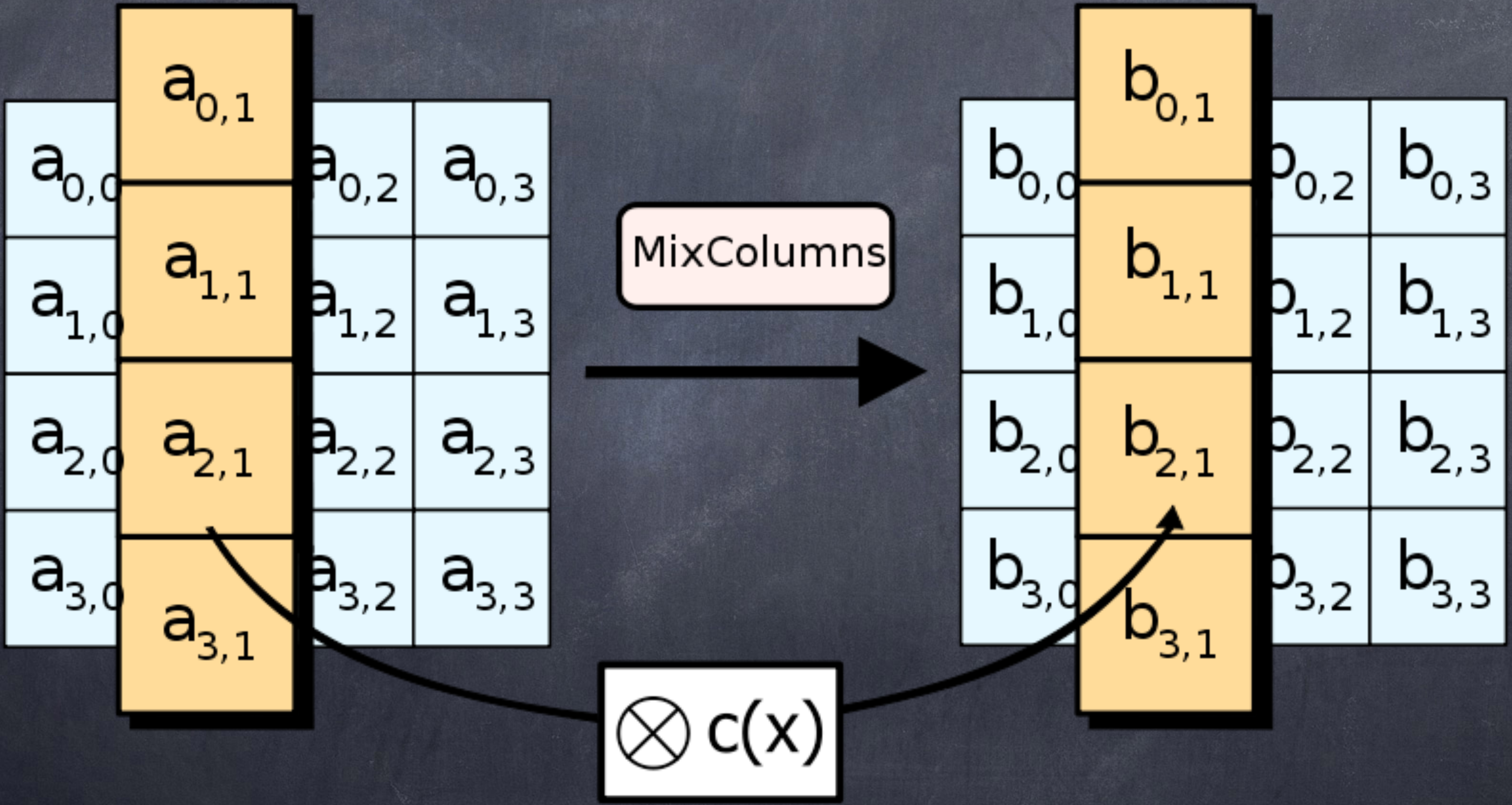
The AES construction

Stage 3 **— ShiftRows**:

In this step, the bytes in each row of the state array are cyclically shifted to the left as follows:

the first row of the array is untouched,
the second row is shifted **one** place to the left,
the third row is shifted **two** places to the left, and
the fourth row is shifted **three** places to the left.
(All shifts are cyclic.)

The AES construction



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes\, c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

The AES construction

Stage 4 — **MixColumns**:

In this step, an invertible linear transformation is applied to each column.

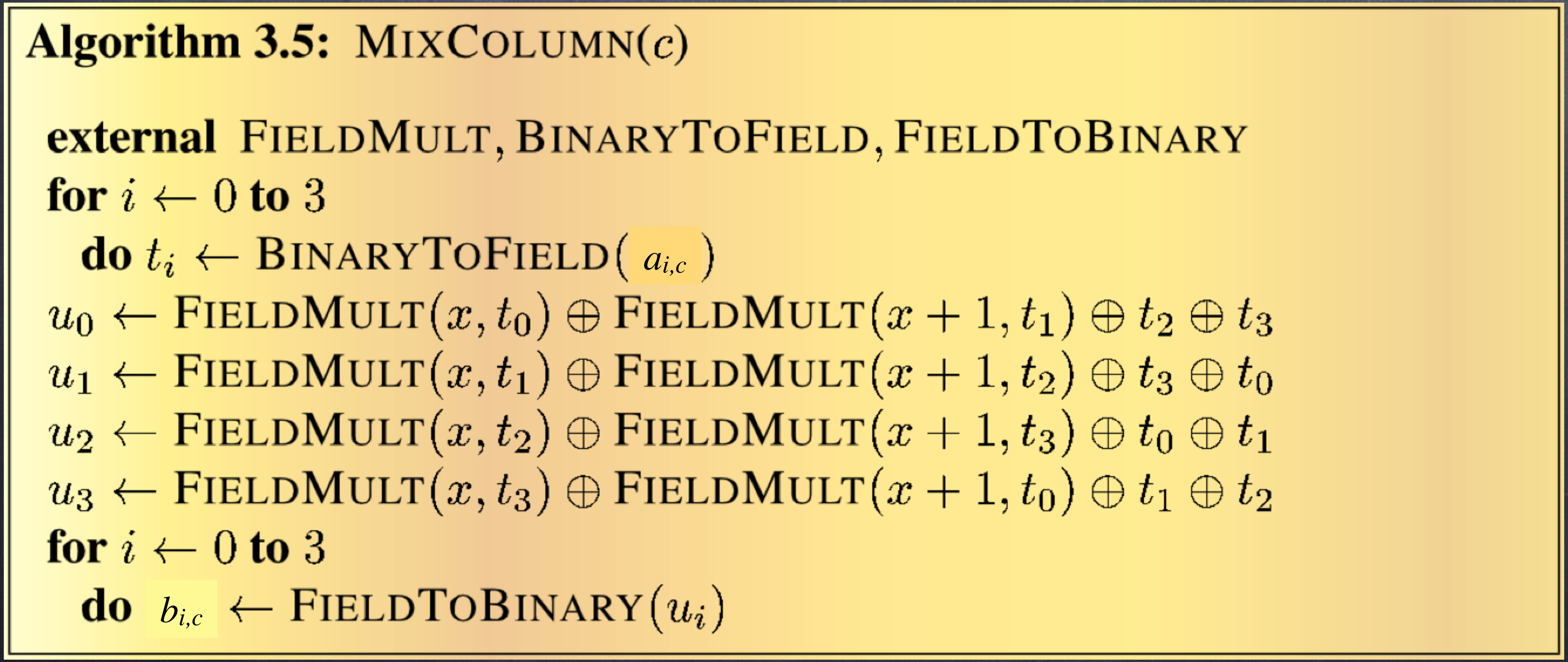
One can think of this as matrix multiplication (over the appropriate field).

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_{0,c} \\ a_{1,c} \\ a_{2,c} \\ a_{3,c} \end{pmatrix} = \begin{pmatrix} b_{0,c} \\ b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{pmatrix}$$

$03_{16} = 00000011_2 = 0x^7+0x^6+0x^5+0x^4+0x^3+0x^2+1x+1 = x+1$

$02_{16} = 00000010_2 = 0x^7+0x^6+0x^5+0x^4+0x^3+0x^2+1x+0 = x$

$01_{16} = 0000001_2 = 0x^7+0x^6+0x^5+0x^4+0x^3+0x^2+0x+1 = 1$



Algorithm 3.5: MIXCOLUMN(c)

```
external FIELDMULT, BINARYTOFIELD, FIELDTOBINARY
for $i \leftarrow 0$ to 3
  do $t_i \leftarrow$ BINARYTOFIELD($a_{i,c}$)
$u_0 \leftarrow$ FIELDMULT($x, t_0$) $\oplus$ FIELDMULT($x+1, t_1$) $\oplus t_2 \oplus t_3$
$u_1 \leftarrow$ FIELDMULT($x, t_1$) $\oplus$ FIELDMULT($x+1, t_2$) $\oplus t_3 \oplus t_0$
$u_2 \leftarrow$ FIELDMULT($x, t_2$) $\oplus$ FIELDMULT($x+1, t_3$) $\oplus t_0 \oplus t_1$
$u_3 \leftarrow$ FIELDMULT($x, t_3$) $\oplus$ FIELDMULT($x+1, t_0$) $\oplus t_1 \oplus t_2$
for $i \leftarrow 0$ to 3
  do $b_{i,c} \leftarrow$ FIELDTOBINARY($u_i$)
```

The AES construction

- By viewing **stages 3** and **4** together as a “mixing” step, we see that each round of **AES** has the structure of a substitution-permutation network:
- The round sub-key is first **XOR**ed with the input to the current round;
- Next, a small, invertible function is applied to “chunks” of the resulting value;
- Finally, the bits of the result are mixed in order to obtain diffusion.

The AES construction

- The only difference is that, unlike our general description of substitution-permutation networks, here the mixing step does not consist of a simple permutation of the bits but is instead carried out using an invertible linear transformation of the bits.
- Simplifying things a bit and looking at a trivial **3**-bit example, an invertible linear transformation might map $\boldsymbol{x}$ to $\langle\, \boldsymbol{x_1} \oplus \boldsymbol{x_2} \parallel \boldsymbol{x_2} \oplus \boldsymbol{x_3} \parallel \boldsymbol{x_1} \oplus \boldsymbol{x_2} \oplus \boldsymbol{x_3} \,\rangle$.

The AES construction

- The number of rounds in **AES** depends on the length of the key.
- There are **10** rounds for a **128**-bit key, **12** rounds for a **192**-bit key, and **14** rounds for a **256**-bit key.
- In the final round of **AES** the **MixColumns** stage is replaced with an additional **AddRoundKey** step.

Security of AES

As we have mentioned, the **AES** cipher was subject to intensive scrutiny during the selection process and this has continued ever since.

To date, the only non-trivial cryptanalytic attacks that have been found are for reduced-round variants of **AES**.

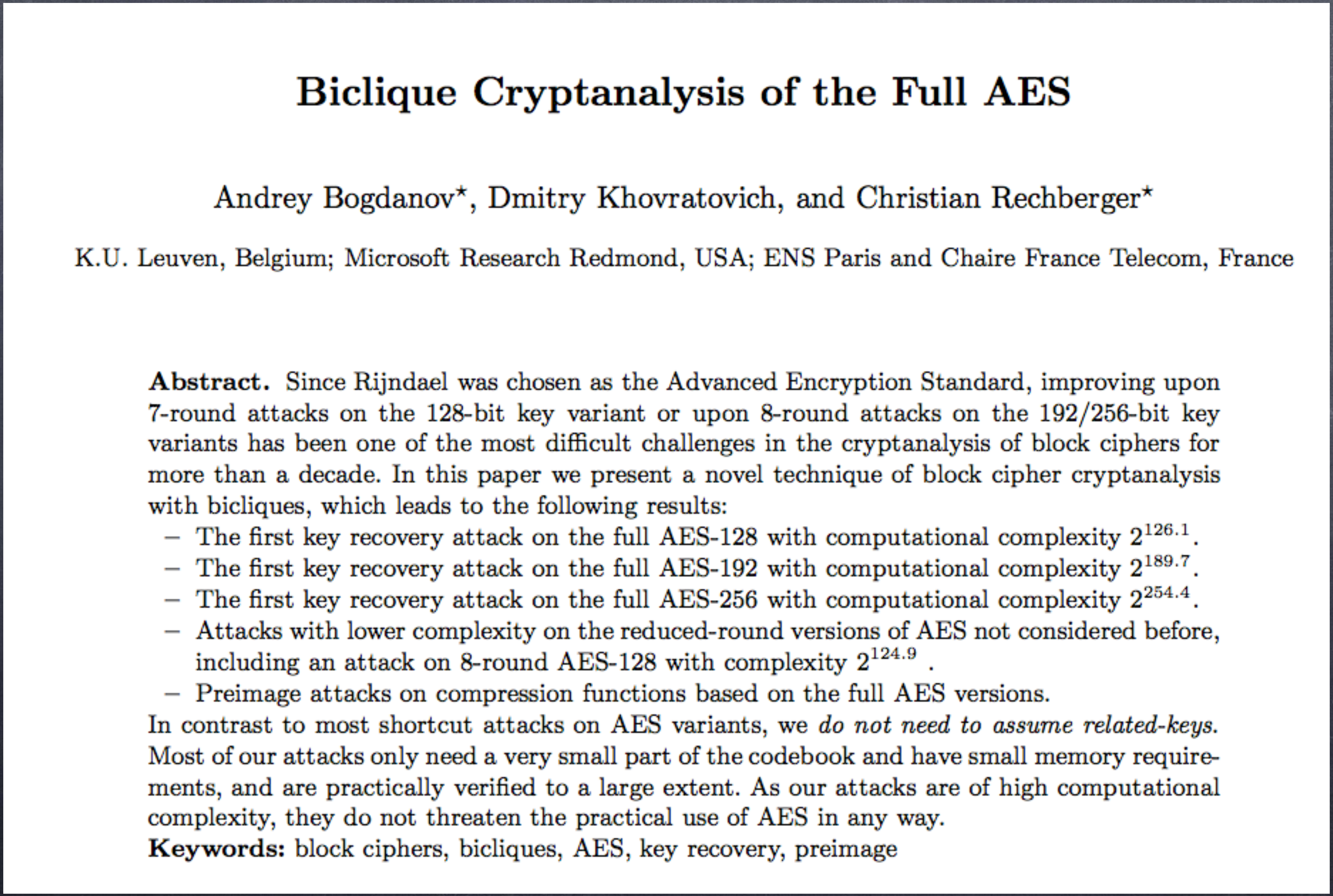
It is often hard to compare cryptanalytic attacks because each tends to perform better with regard to some parameter; we describe the complexity of one set of attacks merely to give a flavor of what is known.

Security of AES

- Known attacks on **6**-round **AES** for **128**-bit keys (using on the order of $\mathbf{2^{72}}$ encryptions), **8**-round **AES** for **192**-bit keys (using on the order of $\mathbf{2^{188}}$ encryptions), and **8**-round **AES** for **256**-bit keys (using on the order of $\mathbf{2^{204}}$ encryptions).

- We stress that the above attacks are for *reduced-round* variants of **AES**, and as of today no attack better than exhaustive key search is known for the full **AES** construction.

Security of AES



Biclique Cryptanalysis of the Full AES

Andrey Bogdanov*, Dmitry Khovratovich, and Christian Rechberger*

K.U. Leuven, Belgium; Microsoft Research Redmond, USA; ENS Paris and Chaire France Telecom, France

Abstract. Since Rijndael was chosen as the Advanced Encryption Standard, improving upon 7-round attacks on the 128-bit key variant or upon 8-round attacks on the 192/256-bit key variants has been one of the most difficult challenges in the cryptanalysis of block ciphers for more than a decade. In this paper we present a novel technique of block cipher cryptanalysis with bicliques, which leads to the following results:

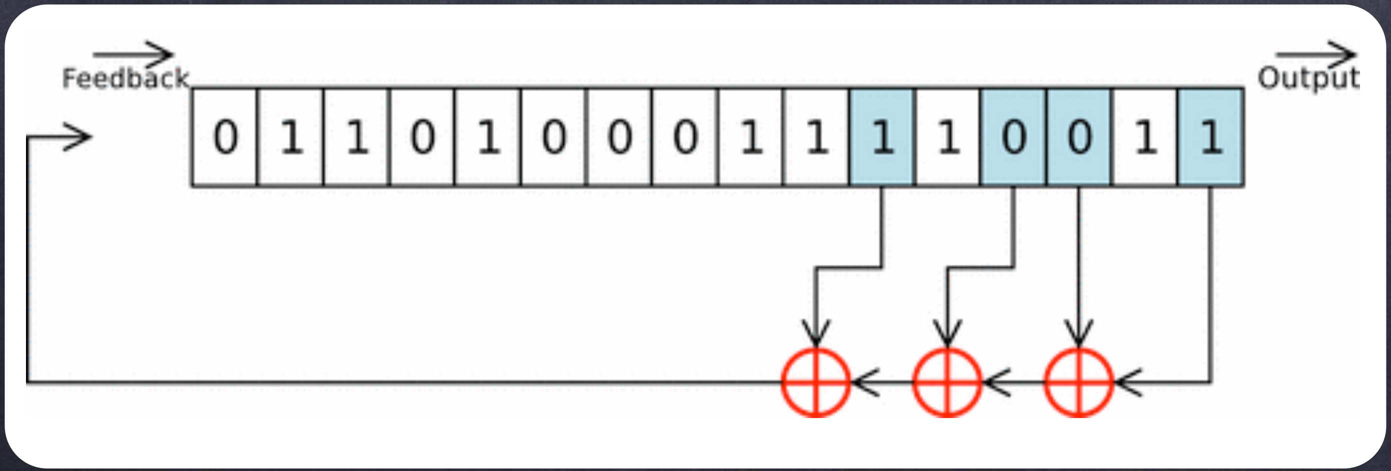
- The first key recovery attack on the full AES-128 with computational complexity $2^{126.1}$.
- The first key recovery attack on the full AES-192 with computational complexity $2^{189.7}$.
- The first key recovery attack on the full AES-256 with computational complexity $2^{254.4}$.
- Attacks with lower complexity on the reduced-round versions of AES not considered before, including an attack on 8-round AES-128 with complexity $2^{124.9}$.
- Preimage attacks on compression functions based on the full AES versions.

In contrast to most shortcut attacks on AES variants, we *do not need to assume related-keys.* Most of our attacks only need a very small part of the codebook and have small memory requirements, and are practically verified to a large extent. As our attacks are of high computational complexity, they do not threaten the practical use of AES in any way.

Keywords: block ciphers, bicliques, AES, key recovery, preimage

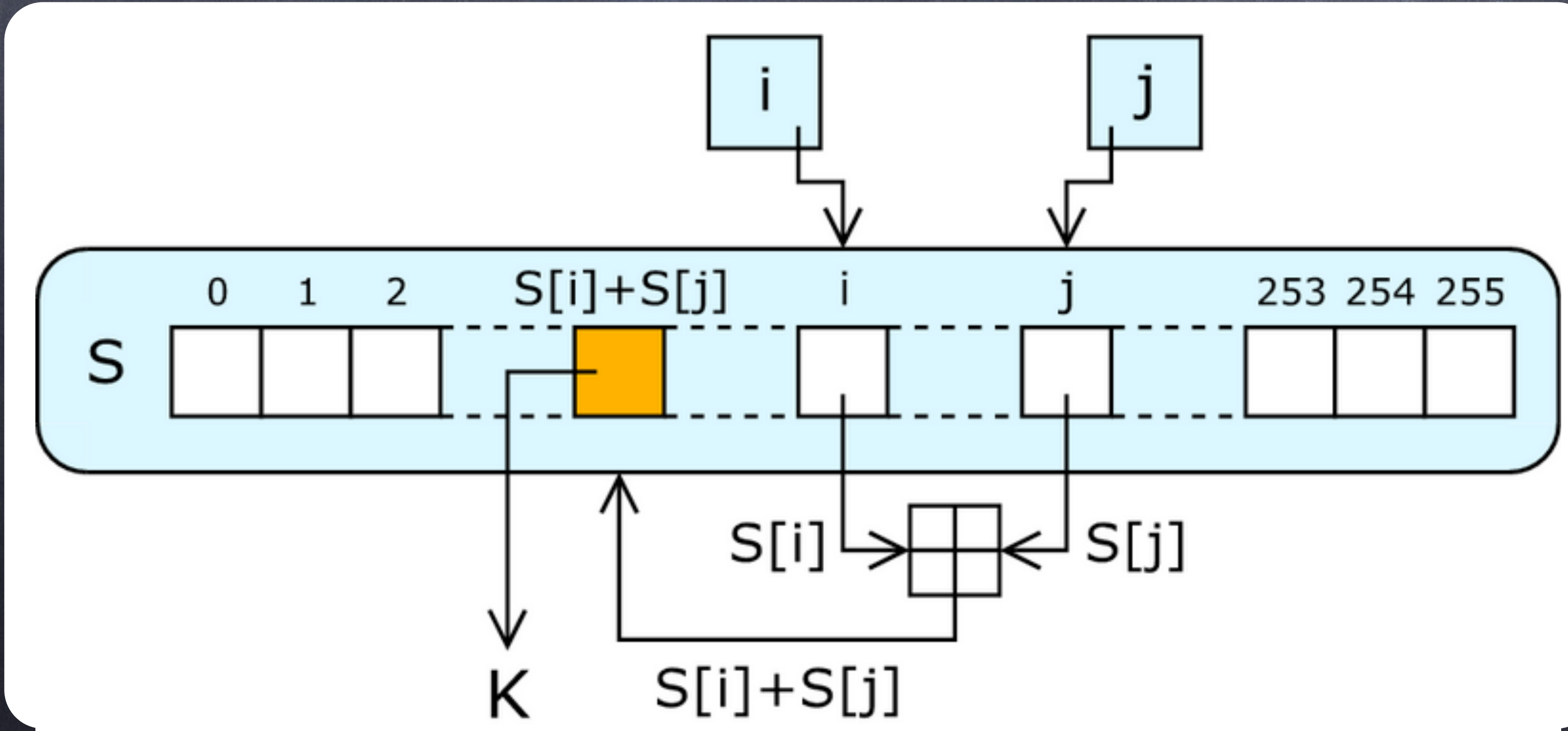
6.1 Practical Constructions of Stream Ciphers

- There are a number of practical constructions of stream ciphers available, and these are typically extraordinarily fast.
- Linear feedback shift registers (**LFSR**s) have, historically, been popular as stream ciphers. However, they have been shown to be horribly insecure (to the extent that the key can be completely recovered given sufficiently-many bytes of the output) and so should never be used today.



Stream Ciphers in Practice: RC4

- A popular example is the stream cipher **RC4** which is widely considered to be secure when used appropriately (see below).
- The security of practical stream ciphers is not yet very well understood, particularly in comparison to *block ciphers*.



Stream ciphers in practice : RC4

- For example, "plain" **RC4** (still widely deployed) is now known to have some significant weaknesses.
- For one, the first few bytes of the output stream generated by **RC4** have been shown to be biased.
- It was also shown that this weakness can be used to feasibly break the **WEP** encryption protocol used in **802.11** wireless networks.
- If **RC4** is to be used, the first **1024** bits or so of the output stream should be discarded.

Stream ciphers in practice?: Blum–Micali

- Let p be a large prime with known factorisation of p-1. Let g be a primitive element mod p.

- Let s_0 be a random seed, $1 \leq s_0 \leq p$. Consider the sequence $s_{i+1} := g^{s_i}$ mod p and the generator

$$G: s_0 \longmapsto \mathrm{half}(s_0) \parallel \mathrm{half}(s_1) \parallel \ldots \parallel \mathrm{half}(s_\ell)$$

where "$\parallel$" stands for concatenation and

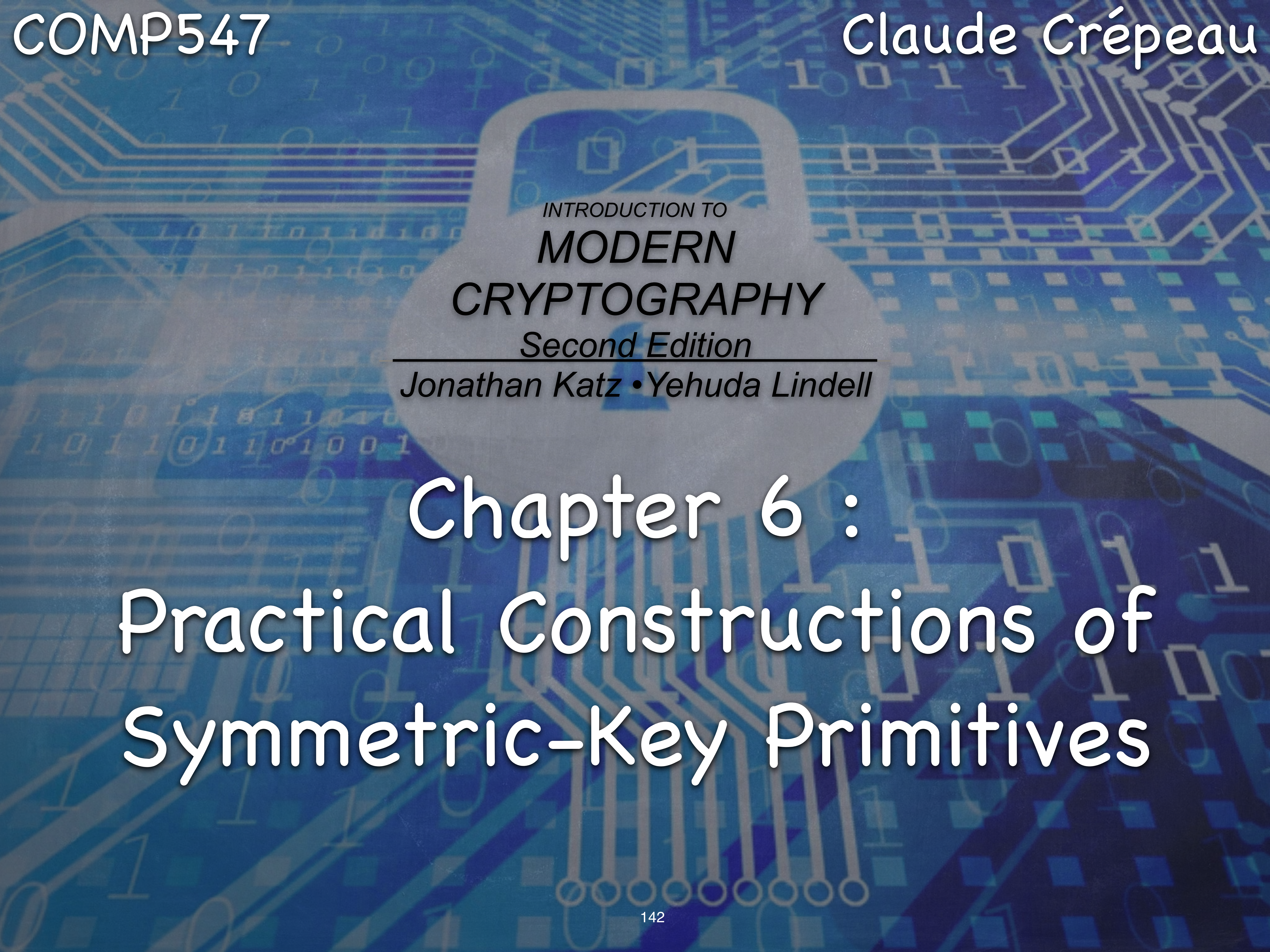
$$\mathrm{half}(x) = \begin{cases} 0 & \text{if } 0 \leq x \leq {}^{(p-1)}/_2 \\ 1 & \text{if } {}^{(p+1)}/_2 \leq x \leq p\text{-}1. \end{cases}$$

- G is pseudorandom unless *discrete log* mod p is easy.

Tasks / Security	Encryption	Authentication	Identification	Quantum
Symmetric Informational	Miller-Vernam One-Time PAD	Wegman-Carter Universal Hash	Simple Solutions	Quantum Key Distribution
Symmetric Computational	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	Quantum Attacks, Q-Safety
Asymmetric Computational	RSA, ElGammal, Blum-Goldwasser	RSA, DSA, etc	Guilloux-Quisquater, Schnor, etc	Quantum Attacks, Q-Safety

🟩 DONE	🟨 IN PROGRESS	⬜ TO DO	🟥 GIVE UP

COMP547 Claude Crépeau



INTRODUCTION TO

MODERN CRYPTOGRAPHY

Second Edition

Jonathan Katz •Yehuda Lindell

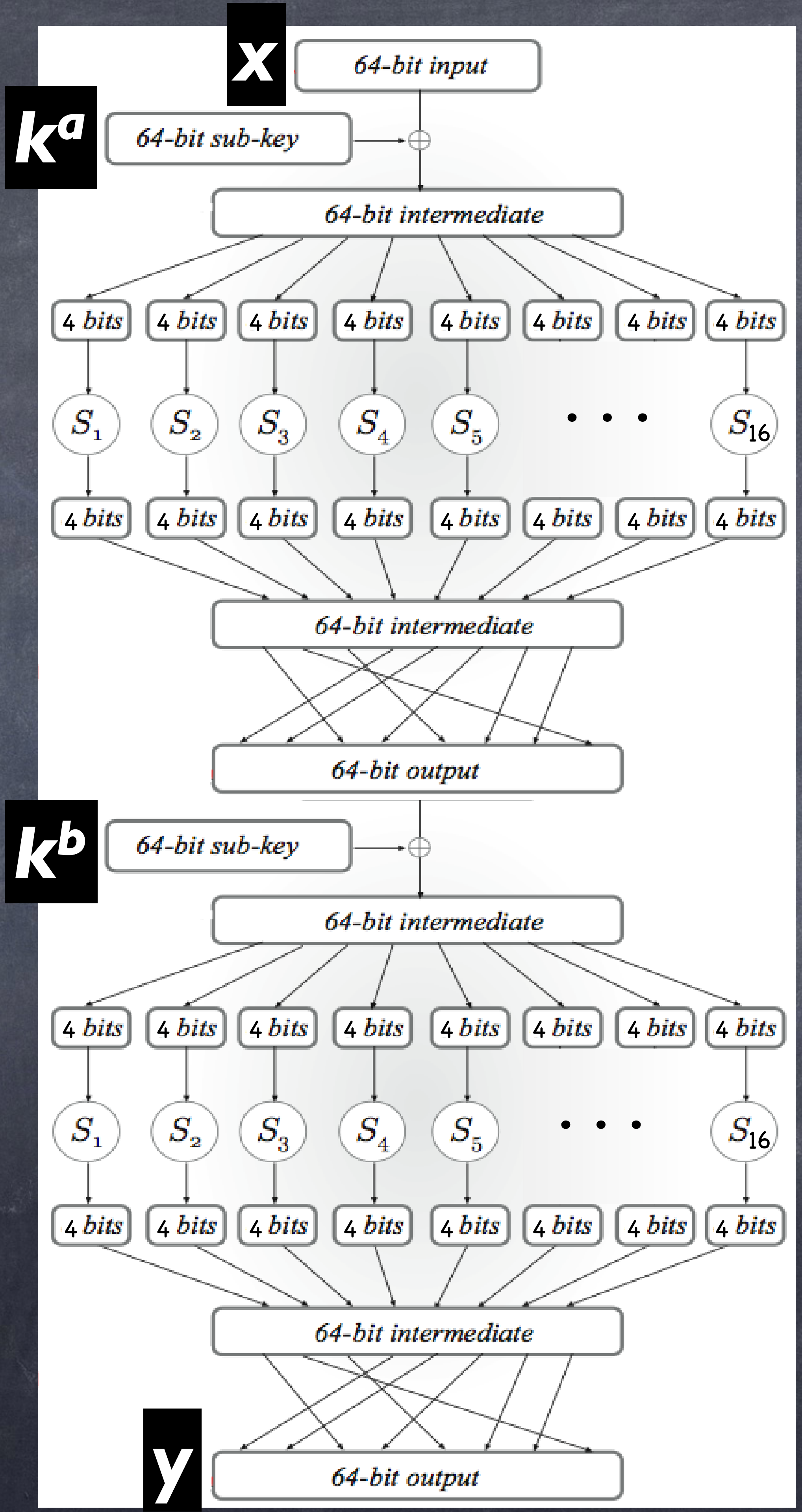
Chapter 6 : Practical Constructions of Symmetric-Key Primitives

Attack on a two-round Subs-Per Network

Let the block length of the cipher be **64** bits, and let each ***S***-box have a **4**-bit input/output length.

Furthermore, let the key $\boldsymbol{k}$ be of length **128** bits where the first half $\boldsymbol{k}^a \in \{\mathbf{0},\mathbf{1}\}^{\mathbf{64}}$ of the key is used in the first round and the second half $\boldsymbol{k}^b \in \{\mathbf{0},\mathbf{1}\}^{\mathbf{64}}$ is used in the second round.

We use independent keys here to simplify the description of the attack below, but this only makes the attack more difficult.

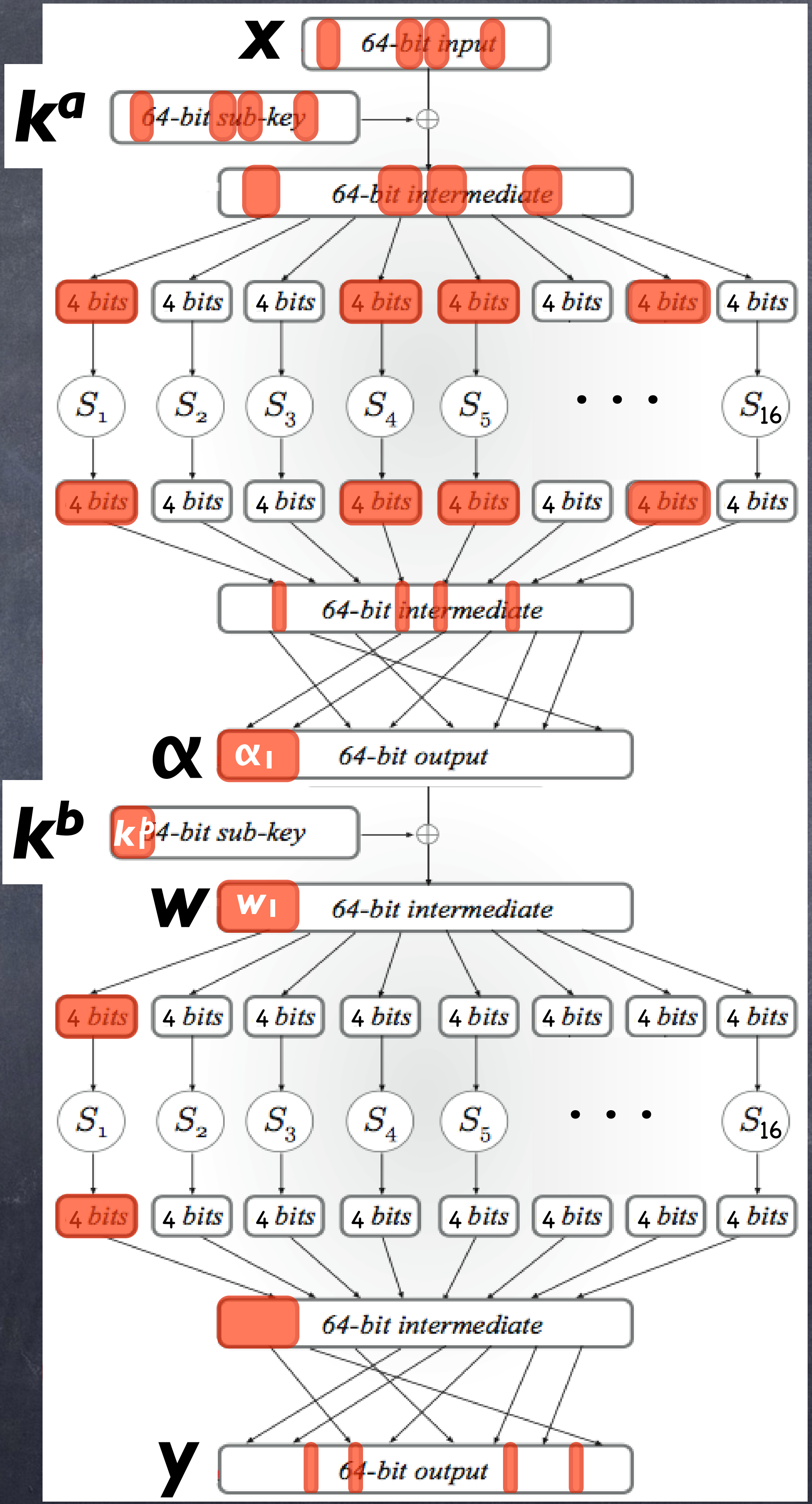

x
64-bit input
k^a
64-bit sub-key
64-bit intermediate
4 bits
S1 S2 S3 S4 S5 . . . S16
4 bits
64-bit intermediate
64-bit output
k^b
64-bit sub-key
64-bit intermediate
4 bits
S1 S2 S3 S4 S5 . . . S16
4 bits
64-bit intermediate
y
64-bit output

Attack on a two-round Subs-Per Network

Say the adversary is given an input $\boldsymbol{x}$ and the output $\boldsymbol{y = F_k(x)}$ of the cipher.

The adversary begins by "working backward", inverting the mixing permutation and $\boldsymbol{S}$-boxes in the second round of the cipher (as in the previous attack).

Denote by $\boldsymbol{w_1}$ the first **4** bits of the result. Letting $\boldsymbol{\alpha_1}$ denote the first **4** bits of the output of the first round, we have that $\boldsymbol{w_1 = \alpha_1 \oplus k_1^b}$, where $\boldsymbol{k_1^b}$ denotes the first **4** bits of $\boldsymbol{k^b}$.


x
64-bit input
k^a
64-bit sub-key
64-bit intermediate
4 bits
S_1
S_2
S_3
S_4
S_5
• • •
S_{16}
64-bit intermediate
α
α_1
64-bit output
k^b
k^b_1
64-bit sub-key
w
w_1
64-bit intermediate
64-bit intermediate
y
64-bit output

Attack on a two-round Subs-Per Network

- The important observation here is that when "working forward" starting with the input ***x***, the value of $\boldsymbol{\alpha_1}$ is influenced by at most **4** different ***S***-boxes (because, in the worst case, each bit of $\boldsymbol{\alpha_1}$ comes from a different ***S***-box in the first round).
- Furthermore, since the mixing permutation of the first round is known, the adversary knows exactly which of the ***S***-boxes influence $\boldsymbol{\alpha_1}$.

Attack on a two-round Subs-Per Network

- This, in turn, means that at most **16** bits of the key $\boldsymbol{k^a}$ (in known positions) influence the computation of these four ***S***-boxes.

- It follows that the adversary can guess the appropriate **16** bits of $\boldsymbol{k^a}$ and the **4**-bit value $\boldsymbol{k_i^b}$, and then *verify* possible correctness of this guess using the known input/output pair **(*x*,*y*)**.

Attack on a two-round Subs-Per Network

- This verification is carried out by **XOR**ing the relevant **16** bits of the input $\boldsymbol{x}$ with the relevant **16** bits of $\boldsymbol{k^a}$, computing the resulting $\boldsymbol{\alpha_1}$, and then computing $\boldsymbol{k_1^b = w_1 \oplus \alpha_1}$.

- Proceeding in this way, the adversary can exhaustively find all values of these **16** bits of the key that are consistent with the given $\boldsymbol{(x, y)}$.

- This takes time $\boldsymbol{2^{16}}$ to try all possibilities.