# MD5

Terec Andrei
Pop David
George Popovici

# What is MD5

The MD5 message-digest algorithm is a hash function producing a 128-bit hash value.

In 1996, a flaw was found in the design of MD5. While it was not deemed a fatal weakness at the time, cryptographers began recommending the use of other algorithms, such as SHA-1, which has since been found to be vulnerable as well.

In 2004 it was shown that MD5 is not collision-resistant.

# How MD5 works

The md5 algorithm does the following steps:

- Padding the input to a multiple of 512 bits
- Process each 512 bits of the message and update an internal state
  - The input is split into 16 32 bits blocks
  - This blocks are processed in 4 rounds for a total of 64 operations

# Padding

- In order to make the input length in bits a multiple of 512, the following process is done:
- Append the 0 bit to the input until the input length is 448 (mod 512)
- Append the original length of the input (mod $2^{64}$)

# Initial state

- The initial state is an hardcoded 128 bits state, that is split into 4 32 bits block called A, B, C, D

State = 0x 67425301 EDFCBA45 98CBADFE 13DCE476

A = 0x67425301

B = 0xEDFCBA45

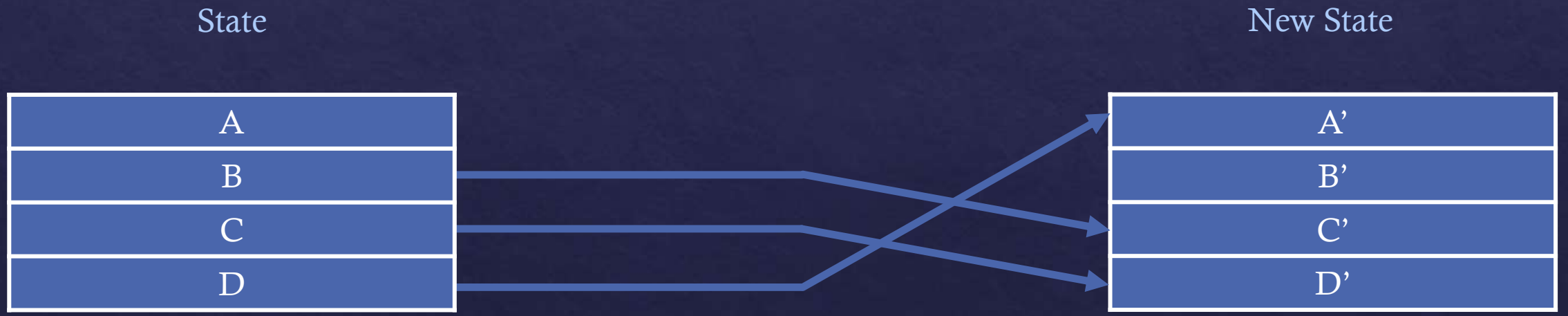
C = 0x98CBADFE

D = 0x13DCE476

# How the first loop works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 0e6c 670f | 2f27 1acd | 1b3c 30c5 | f17c a0b5 | 6031 b5a0 | 949f f572 | 9ed8 28d5 | d898 799a | 7051 0ffe | 7e81 ae3e | c3c3 0d84 | e54a 0cc4 | d3e5 5b37 | 1cbb acb7 | 3d08 8dca | b8f3 5cb8 |

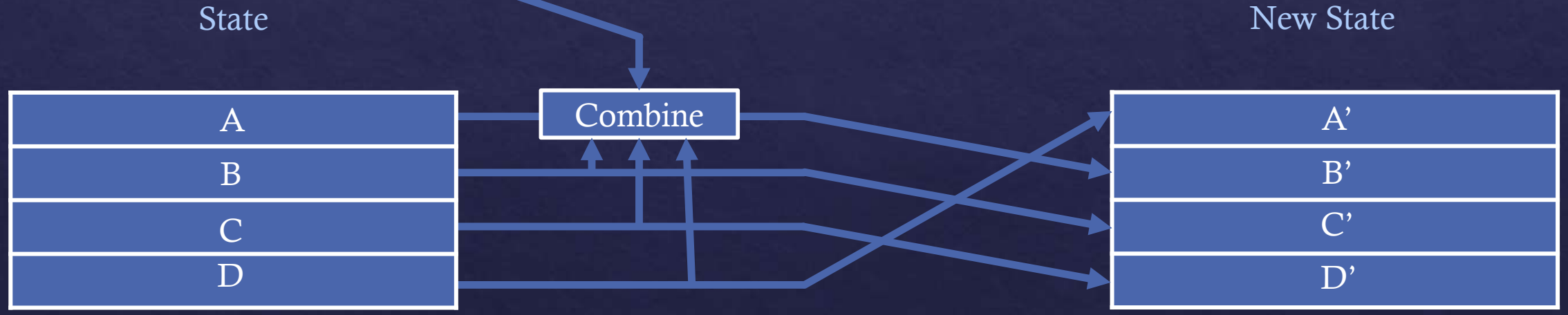
i=0
i%16=0



# How the first round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 0e6c 670f | 2f27 1acd | 1b3c 30c5 | f17c a0b5 | 6031 b5a0 | 949f f572 | 9ed8 28d5 | d898 799a | 7051 0ffe | 7e81 ae3e | c3c3 0d84 | e54a 0cc4 | d3e5 5b37 | 1cbb acb7 | 3d08 8dca | b8f3 5cb8 |

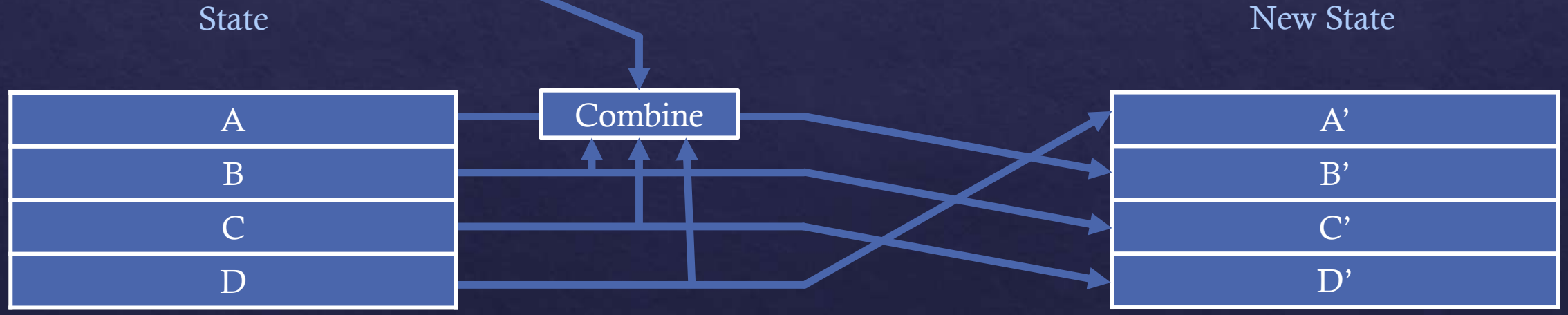
i=0
i%16=0



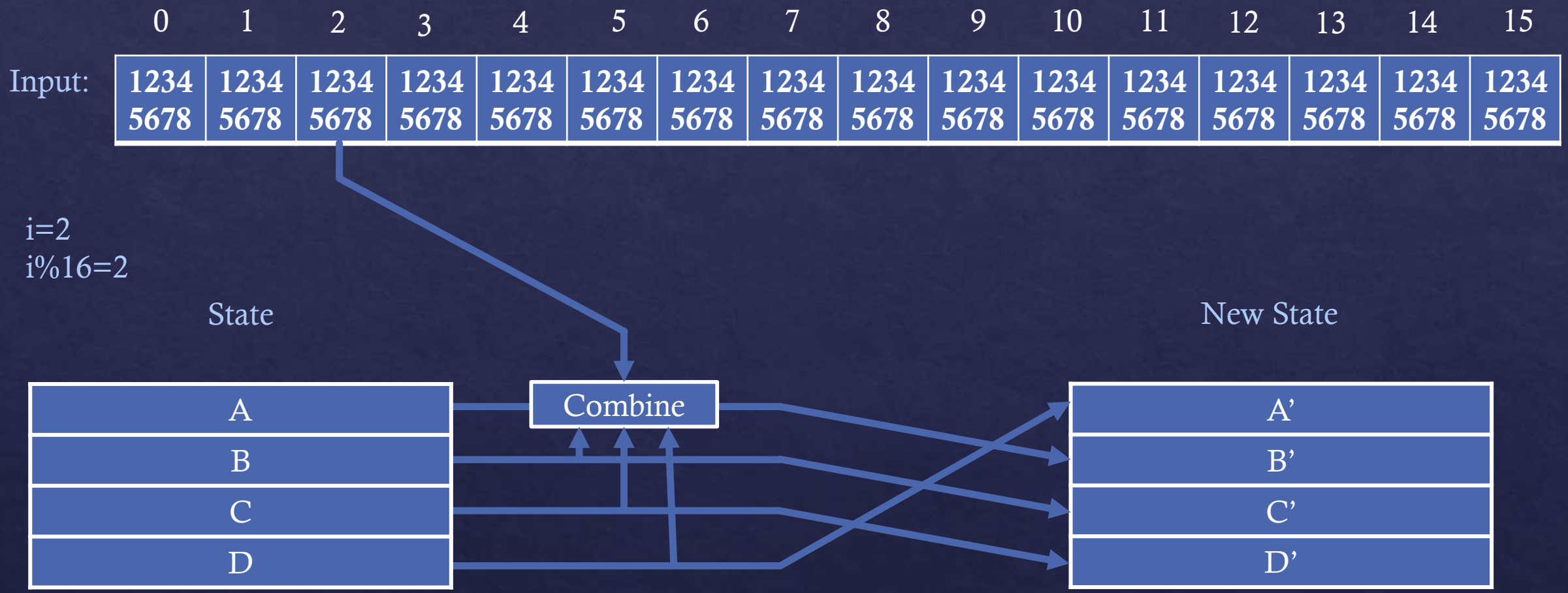
# How the first round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 |

i=1
i%16=1



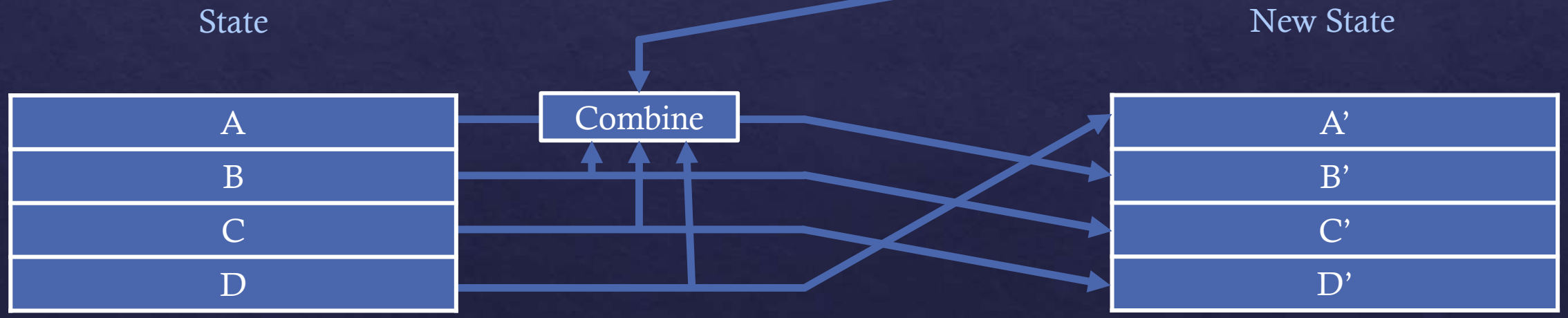
# How the first round works



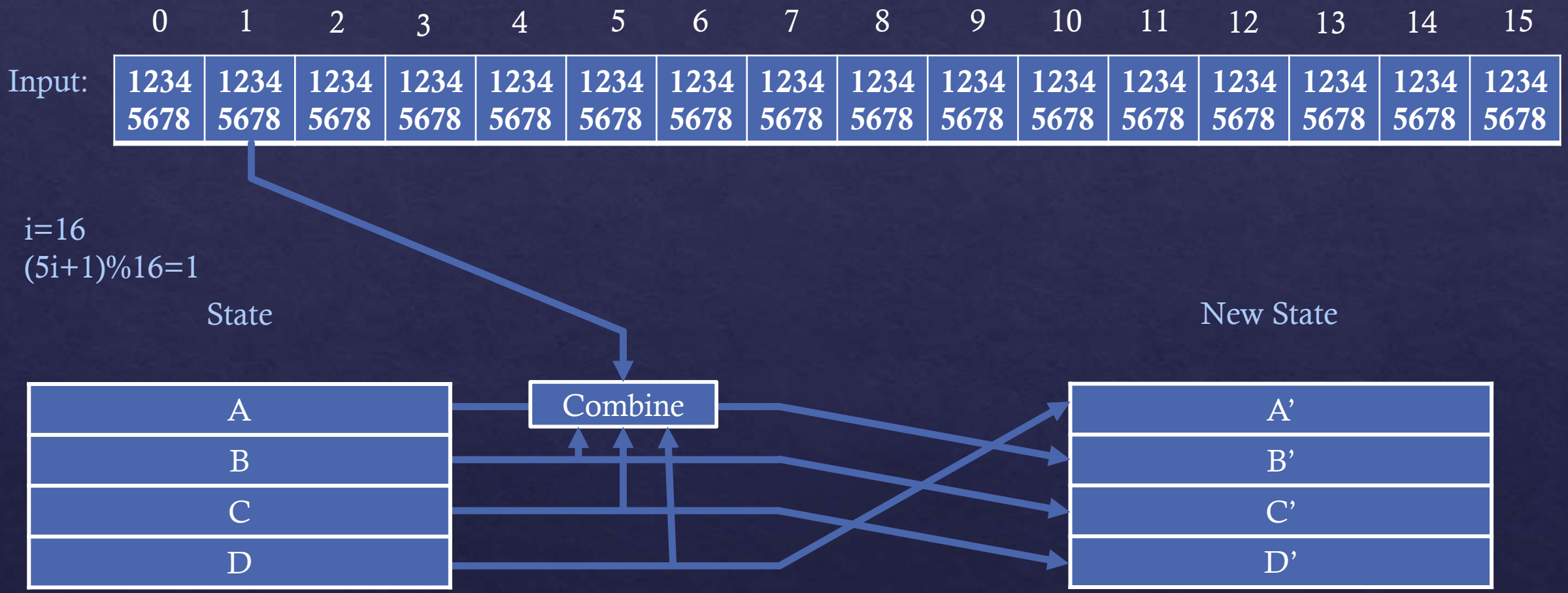
# How the first round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 |

i=15
i%16=15



# How the second round works

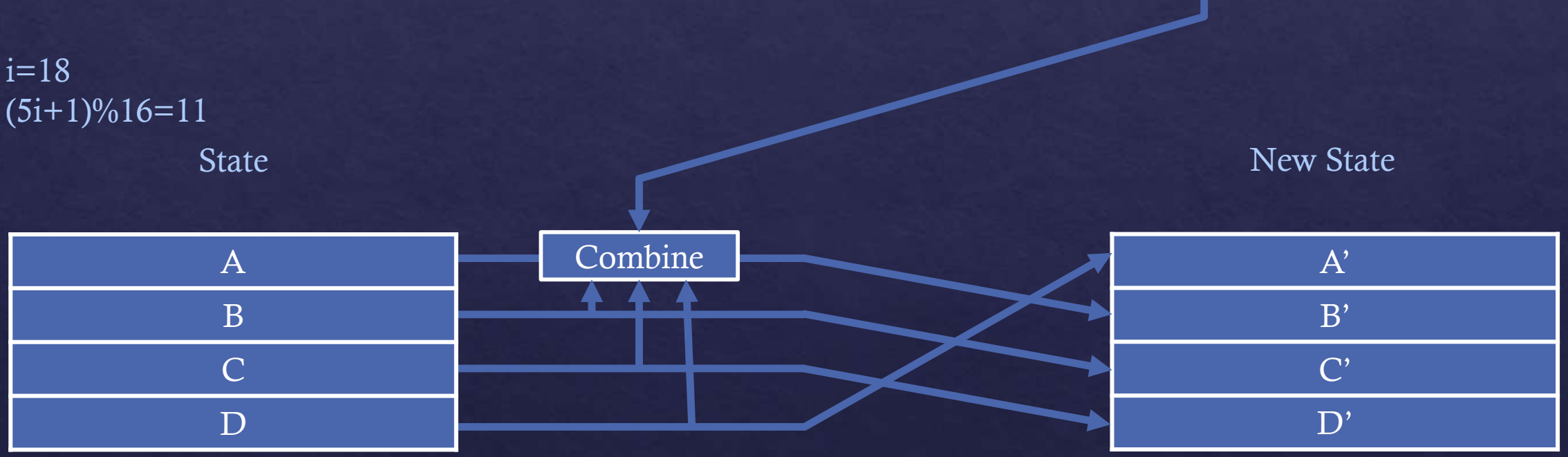


# How the second round works

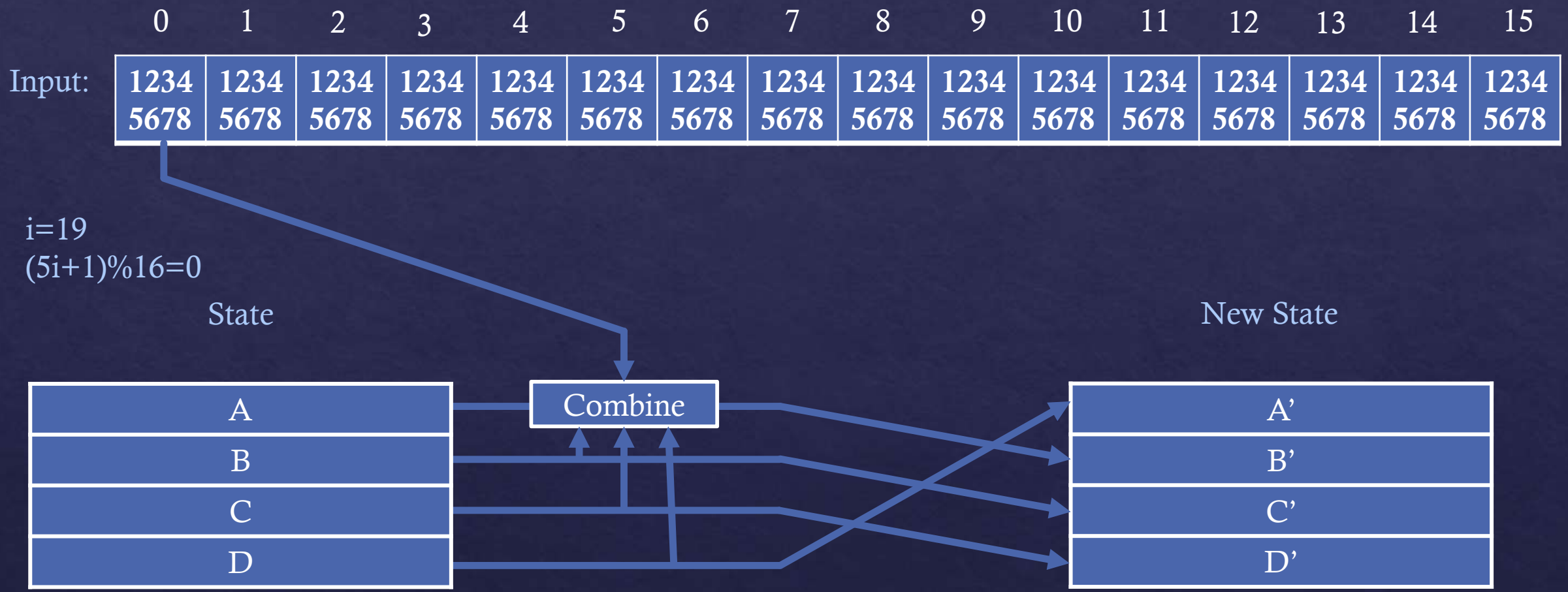
# How the second round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 |

i=18
(5i+1)%16=11



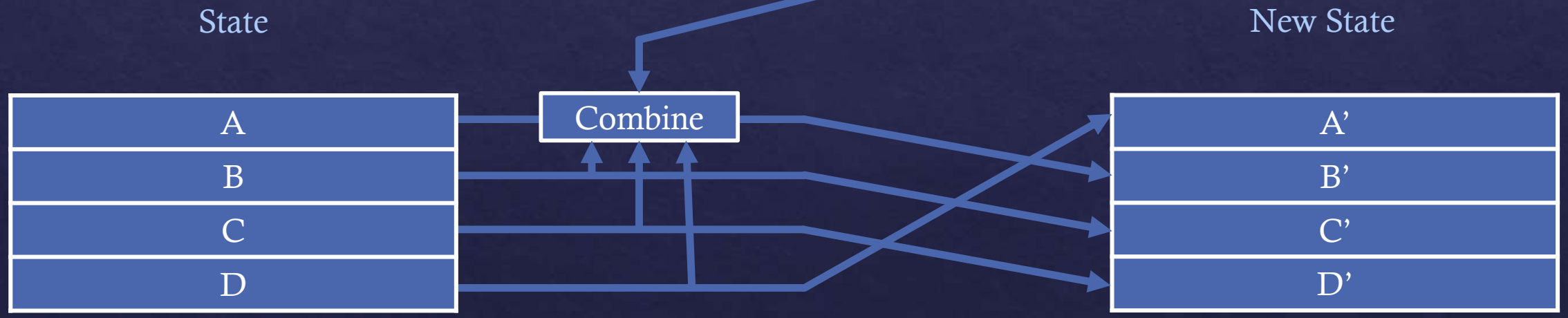
# How the second round works



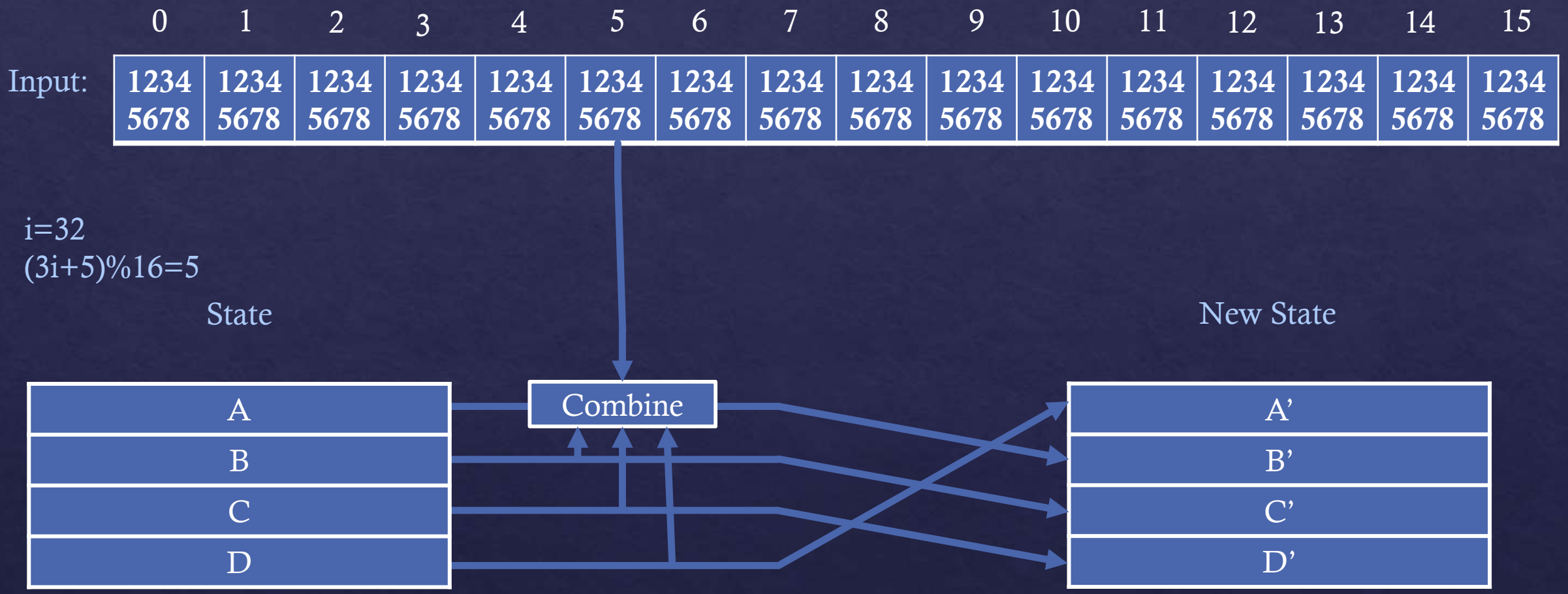
# How the second round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 |

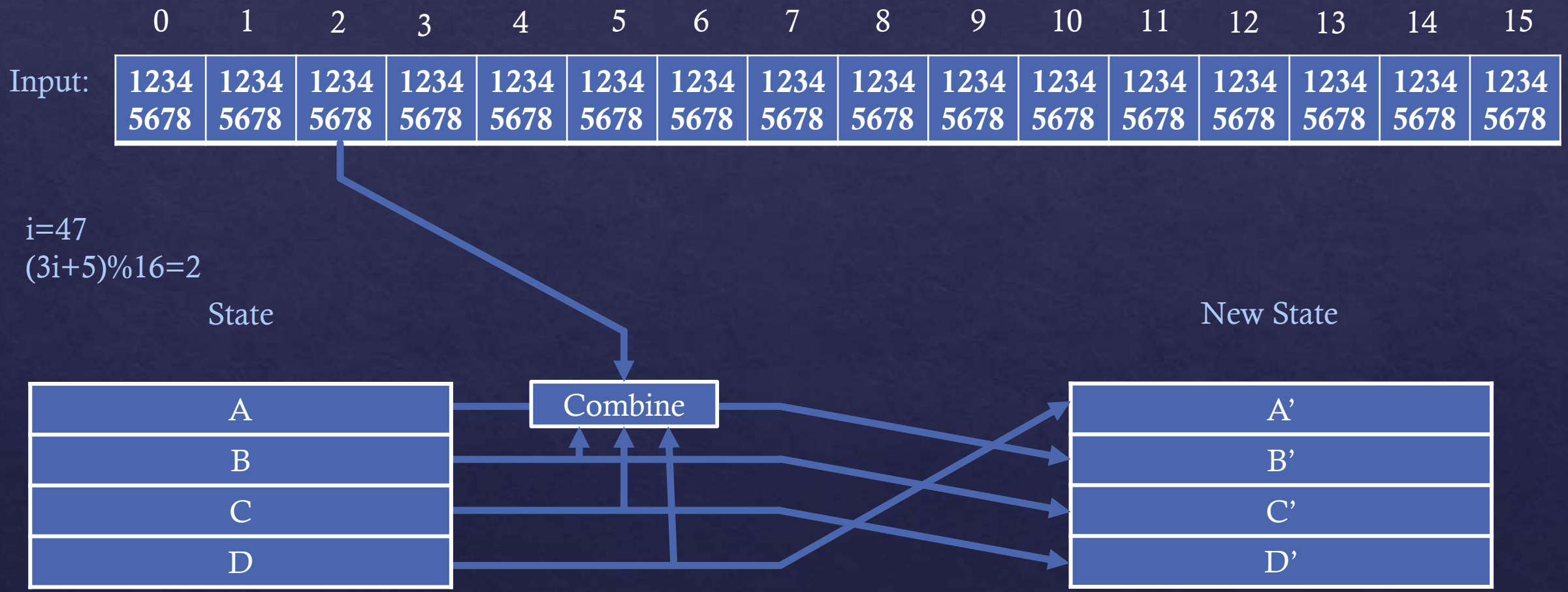
i=31
(5i+1)%16=12



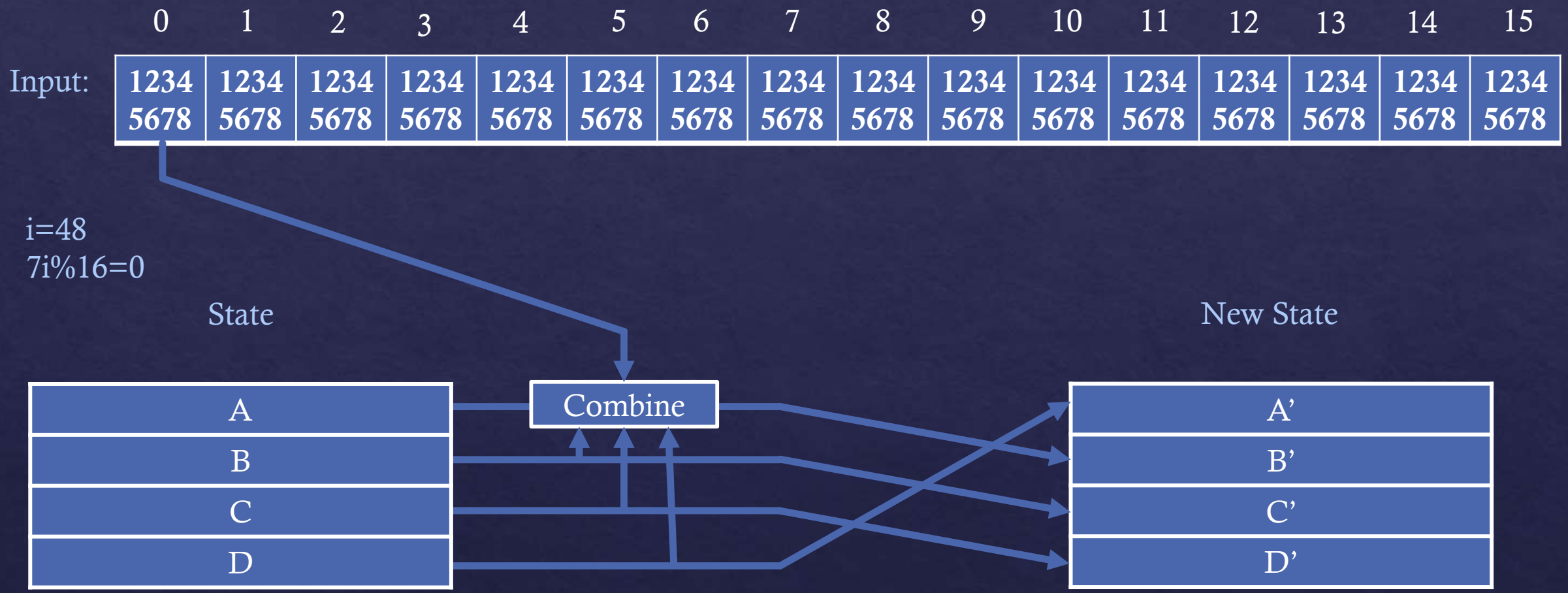
# How the third round works



# How the third round works



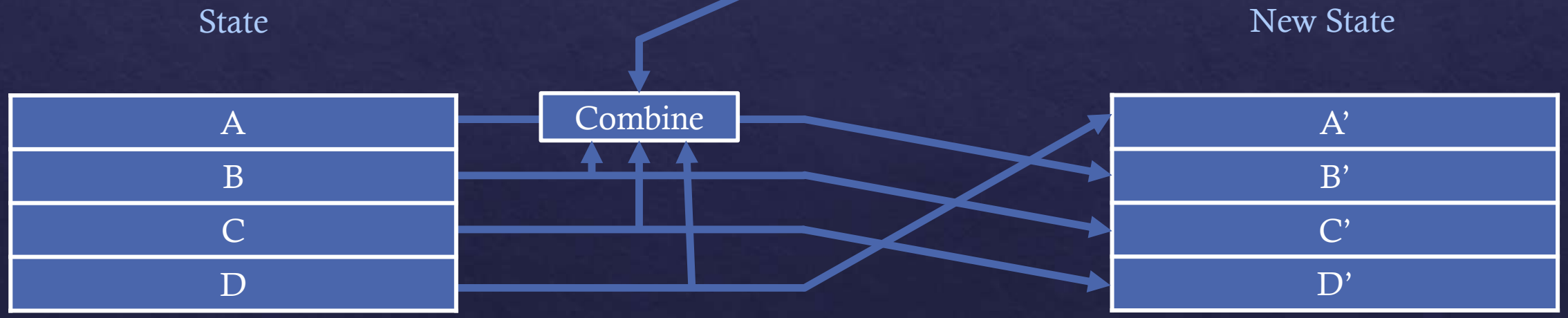
# How the forth round works



# How the forth round works

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input: | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 | 1234 5678 |

i=63
7i%16=9



# The final result

We concatenate the final state back to get the 128 bits hash.

State =

| A | B | C | D |
|---|---|---|---|

# The combine function

Combine(A, B, C, D, Input, i) = transform(A + F(B, C, D, i) + Input[currentIndex(i)], B, i)

- The addition is done modulo 2^32

$$\text{currentIndex}(i) = \begin{cases} I \,\%\, 16 & , 0 \le i < 16 \quad \text{(First round)} \\ (5i + 1) \,\%\, 16 & , 16 \le i < 32 \quad \text{(Second round)} \\ (3i + 5) \,\%\, 16 & , 32 \le i < 48 \quad \text{(Third round)} \\ 7i \,\%\, 16 & , 48 \le i < 64 \quad \text{(Forth round)} \end{cases}$$

$$F(B, C, D, i) = \begin{cases} (B \,\&\, C) \mid ((\sim B) \,\&\, D) & , 0 \le i < 16 \quad \text{(First round)} \\ (D \,\&\, B) \mid ((\sim D) \,\&\, C) & , 16 \le i < 32 \quad \text{(Second round)} \\ B \,\hat{}\, C \,\hat{}\, D & , 32 \le i < 48 \quad \text{(Third round)} \\ C \,\hat{}\, (B \mid (\sim D)) & , 48 \le i < 64 \quad \text{(Forth round)} \end{cases}$$

# The combine function

transform(input, B, i) = rotate(input + K[i], r[i]) + B

The rotate(input, r) function rotates left the bits in input by r positions

K[i] and r[i] are 64 constants defined in the md5 algorithm

K = [0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501, … ]
r = [7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22, 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9,  … ]

# The combine function

Putting everything together

Combine(A, B, C, D, Input, i) = rotate(A + F(B, C, D, i) + Input[currentIndex(i)] + K[i], r[i]) + B

# Flame malware

- Flame, also known as Flamer, sKyWIper, and Skywiper, is modular computer malware discovered in 2012 that attacks computers running the Microsoft Windows operating system. The program is used for targeted cyber espionage in Middle Eastern countries.
- Budapest University of Technology and Economics - “certainly the most sophisticated malware we encountered during our practice; arguably, it is the most complex malware ever found”
- Flame managed to penetrate numerous computers across the Middle East by falsifying an authentic Microsoft security certificate

# Flame's relation to MD5

- The malware authors identified a Microsoft Terminal Server Licensing Service certificate that inadvertently was enabled for code signing and that still used the weak MD5 hashing algorithm, then produced a counterfeit copy of the certificate that they used to sign some components of the malware to make them appear to have originated from Microsoft.
- Certificate itself had various irregularities, such as no Certificate Revocation List (CRL) Distribution Point (CDP) extension, Authority Information Access (AIA) extension, or a "Microsoft Hydra" critical extension.
- The Microsoft Hydra extension is marked as "critical" and this is crucial to why the attacker needed to perform a collision attack.

# Chosen prefix collision attack

- The attacker can choose two arbitrarily different documents, and then append different calculated values that result in the whole documents having an equal hash value
- Mathematically stated, given two different prefixes p1, p2, the attack finds two suffixes s1 and s2 such that hash(p1 ‖ s1) = hash(p2 ‖ s2) (where ‖ is the concatenation operation).

# MD5 Collision

- The insecurity of the MD5 hash algorithm was demonstrated through a significant discovery made by Xiaoyun Wang and Hongbo Yu, described in their 2004 paper. The linked article provides an accessible summary of their work and explains how it led to practical demonstrations of MD5's weaknesses.
- The researchers showcased a method to generate two different messages that produce the same MD5 hash—a phenomenon known as a collision. This contradicts one of the core properties of cryptographic hash functions, which is resistance to collisions.
- Wang and Yu's method relied on a deep understanding of MD5's internal structure and weaknesses. They exploited vulnerabilities in MD5’s message compression function, specifically targeting the way the algorithm processes input blocks in a series of iterations. By manipulating intermediate values and using mathematical techniques to predict changes, they could efficiently craft pairs of inputs with identical hashes.

# MD5 Collision

d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325**7**1415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70

And

d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325**f**1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70

Each of these blocks has MD5 hash 79054025255fb1a26e4bc422aef54eb4

# MD5 Collision

- The method of Wang and Yu makes it possible, for a given initialization vector s, to find two pairs of blocks M,M' and N,N', such that f(f(s, M), M') = f(f(s, N), N'). It is important that this works for any initialization vector s, and not just for the standard initialization vector s0.
- Combining these observations, it is possible to find pairs of files of arbitrary length, which are identical except for 128 bytes somewhere in the middle of the file, and which have identical MD5 hash. Indeed, let us write the two files as sequences of 64-byte blocks:

  $M_0, M_1, \ldots, M_{i-1}, M_i, M_{i+1}, M_{i+2}, \ldots, M_n,$

  $M_0, M_1, \ldots, M_{i-1}, N_i, N_{i+1}, M_{i+2}, \ldots, M_n.$

- The blocks at the beginning of the files, $M_0, \ldots, M_{i-1}$, can be chosen arbitrarily. Suppose that the internal state of the MD5 hash function after processing these blocks is $s_i$.

# MD5 Collision

- Now we can apply Wang and Yu's method to the initialization vector $s_i$, to find two pairs of blocks $M_i$, $M_{i+1}$ and $N_i$, $N_{i+1}$, such that
- $s_{i+2} = f(f(s_i, M_i), M_{i+1}) = f(f(s_i, N_i), N_{i+1})$.
- This guarantees that the internal state $s_{i+2}$ after the $i$+2st block will be the same for the two files. Finally, the remaining blocks $M_{i+2}$, …, $M_n$ can again be chosen arbitrarily.
- So how can we use this technique to produce a pair of programs (or postscript files) that have identical MD5 hash, yet behave in arbitrary different ways? This is simple. All we have to do is write the two programs like this:
- Program 1: if (data1 == data1) then { good_program } else { evil_program }
- Program 2: if (data2 == data1) then { good_program } else { evil_program }
- and arrange things so that "data1" = $M_i$, $M_{i+1}$ and "data2" = $N_i$, $N_{i+1}$ in the above scheme. This can even be done in a compiled program, by first compiling it with dummy values for data1 and data2, and later replacing them with the properly computed values.

# Bibliography


- Flame malware collision attack explained - Security Research & Defense - Site Home - TechNet Blogs (archive.org)
- Wikipedia
- MD5 collision