PEM-encoded key/certificate

BEGIN ... END envelope of a PEM file.

```
-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----
```

Token-by-token breakdown

Token	Meaning
`-----BEGIN`	literal text “-----BEGIN ”
`[A-Z ]+`	one or more: any of: uppercase letters, “ ”
`-----`	literal text “-----”
`[\s\S]+?`	one or more (lazy — as few as possible): any of: whitespace, non-whitespace
`-----END`	literal text “-----END ”
`[A-Z ]+`	one or more: any of: uppercase letters, “ ”
`-----`	literal text “-----”

Quick usage in 7 languages

JavaScript

```
const re = /-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----/;
re.test(value);
```

Python

```
import re
re.match(r"-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----", value)
```

Java

```
Pattern.compile("-----BEGIN [A-Z ]+-----[\\s\\S]+?-----END [A-Z ]
+-----").matcher(value).matches();
```

C# / .NET

```
Regex.IsMatch(value, @"-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----");
```

Go

```
regexp.MustCompile(`-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]
+-----`).MatchString(value)
```

Ruby

```
/-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----/.match?(value)
```

PHP

```
preg_match('~-----BEGIN [A-Z ]+-----[\s\S]+?-----END [A-Z ]+-----~', $value);
```

Common pitfalls

- **Not anchored.** Without ^ and $ this can match a substring anywhere in the input — add anchors if you need the whole value to conform.
- **Uppercase only.** Ranges like [A-Z] won't match lowercase. Add a-z (or the i flag) if lowercase input should be accepted.
- **Escape it correctly per language.** In Java and JavaScript strings each backslash must be doubled (\\d); in Python, Go, and C# use raw/verbatim strings so the backslashes survive.
- **Validate beyond format.** Matching the format doesn't guarantee the value is real. Confirm the pem-encoded key/certificate against a source of truth (database, API, or checksum) where it matters.